UNIVERSIDADE FEDERAL DE PELOTAS
FACULDADE DE METEOROLOGIA
DEPARTAMENTO DE METEOROLOGIA

# Introdução ao *NCAR Command Language* (NCL), com aplicações em Meteorologia

Apostila gerada por meio do projeto de ensino *Elaboração de um Guia em Português do NCAR Command Language com aplicações em Meteorologia*, cadastrado na Pró-Reitoria de Graduação da Universidade Federal de Pelotas sob o número 002.2011.

*Mateus da Silva Teixeira, Vanúcia Schumacher Pogorzelski,*
*Stefane Fonseca Freitas e Carina Klug Padilha Reinke*

Novembro, 2011

# Prefácio

Esta apostila busca passar aos seus leitores uma breve introdução ao *NCAR Command Language*, um pacote de processamento e visualização de dados geofísicos desenvolvido pelo *National Centers for Atmospheric Research*, dos Estados Unidos.

A elaboração deste livro-apostila é baseada fortemente nos manuais originais do NCL, em PDF. Tanto que as suas estruturas de capítulos são muito similares. Entretanto, como o objetivo deste material é oferecer um suporte mais didático, especialmente ao usuário iniciante, procurou-se oferecer ao longo do texto exemplos mais simples, que permitam o uso rápido e fácil das potencialidades desta linguagem.

É impossível abordar toda a potencialidade no NCL em algumas páginas, o que torna essencial ao usuário visitas constantes ao site oficial do NCL (`http://www.ncl.ucar.edu`) e a participação nas listas de discussão.

Algumas situações mais especiais, mas que despendem uma dissertação maior que aquela proposta no corpo principal do texto, são apresentadas nos apêndices. Situações como acentuação de texto e mapas com divisão política para o Brasil são exemplos do que será encontrado nos apêndices.

Os autores gostariam de contar com a ajuda dos leitores na constante melhora deste livro-apostila, seja com a indicação de problemas com o texto e exemplos quanto com sugestões para os mesmos. As colaborações podem ser enviadas diretamente aos autores, por meio do endereço eletrônico mateus.teixeira@ufpel.edu.br.

# Agradecimentos

Os autores desta apostila gostariam de agradecer ao Sr. Dennis Shea e a Sra. Mary Haley por terem gentilmente permitido a confecção de uma "tradução" dos manuais para o Português e pela ajuda provida durante os últimos anos na realização de tarefas com o NCL.

# Sumário

# Capítulo 1

# Introdução

O estudo dos fenômenos meteorológicos está fortemente ligado a análise de dados observacionais. O conhecimento das variações de temperatura, umidade e pressão, bem como de suas distribuições espaciais permite uma descrição detalhada dos sucessivos estados da atmosfera terrestre e, portanto, da sua evolução no tempo.

Nas primeiras investigações da atmosfera, os dados eram plotados manualmente em mapas geográficos. Uma vez plotados, os mapas meteorológicos eram analisados pelos meteorologistas durante a confecção das previsões meteorológicas. A demora na obtenção dos dados, a existência de erros provocados por dificuldades na sua transmissão entre os pontos geradores e os centros de análise, unidos ao tempo gasto na plotagem manual impossibilitavam a geração de diagnósticos do tempo, em maior frequência.

Com o advento dos computadores, o processo de plotagem foi se tornando cada vez mais rápido. Algoritmos mais complexos e eficientes foram sendo desenvolvidos com o objetivo de se obter descrições espaciais das variáveis meteorológicas com melhores qualidades que aquelas feitas manualmente.

Hoje em dia, a inspeção manual se faz mais importante na etapa de análise das cartas meteorológicas do que na verificação da qualidade dos gráficos gerados. É possível afirmar, inclusive, que a maioria senão a totalidade das cartas plotadas hoje em dia são feitas com a ajuda de computadores.

Caminhando no mesmo ritmo está o processamento dos dados. Enquanto que nos primórdios da meteorologia as calculadoras manuais eram usadas, tornando a tarefa árdua, cansativa e suscetível a erros, nos tempos atuais o

computador se tornou uma peça indispensável na análise e interpretação dos dados meteorológicos.

Há vários softwares que oferecem ferramentas para a análise e plotagem de dados meteorológicos. Cada um com as suas vantagens e desvantagens. A Linguagem de Comandos do NCAR (*NCAR Command Language*, NCL) se destaca por apresentar, num único pacote computacional, ferramentas que possibilitam o acesso a vários formatos de arquivos de dados, a realização de cálculos estatísticos e físicos e que oferecem a plotagem de gráficos com excelente qualidade, particularmente importante à geração e publicação de artigos, trabalhos, livros etc.

O NCL é uma linguagem de programação interpretada, especificamente designada para acessar, analisar e visualizar dados. Possui muitas características comuns a de outras linguagens de programação modernas, incluindo tipos de dados, variáveis, operadores, expressões, declarações condicionais, controladores de fluxo, além de possibilitar a criação de funções e subrotinas. É um software gratuito, com grande parte de seu código livre, e está disponível para sistemas Linux/Unix, MacOSX e Windows (Cygwin).

## 1.1 Instalação

Os desenvolvedores do NCL recomendam a instalação dos arquivos binários executáveis, disponíveis para download no site do NCL. Entretanto, estes arquivos estão disponibilizados apenas para algumas versões dos compiladores GCC e de distribuições do Linux. Assim, poder-se-á necessário instalar o NCL a partir do seu código fonte.

Ao usuário deste guia é recomendada a leitura cuidadosa das instruções de instalação disponíveis no site *Install NCL/NCAR Graphics*, para a instalação dos arquivos binários disponíveis, e no site *How to build and install NCAR Graphics and the NCAR Command Language*, para a instalação do NCL a partir do seu código fonte.

Abaixo são mostrados, rapidamente, os passos para a instalação dos arquivos binários em sistemas Linux, mas antes de instalá-lo deve-se obter algumas importantes informações a respeito da distribuição Linux usada:

- Obter o nome do hardware da máquina usada, digitando, num terminal, o comando **uname -m**. Exemplo: *i686*.

- Obter o sistema operacional usado, digitando, num terminal, o comando **uname -s**. Exemplo: *Linux*.
- Obter a versão do GCC usado, digitando, num terminal, o comando **gcc -v**. Exemplo: *4.3.2*.

Na posse destas informações, é possível baixar - do site http://www.earthsystemgrid.org[1] - a versão do NCL apropriada para o sistema, que, a partir das informações acima obtidas, seria **ncl_ncarg-5.2.1.Linux_i686_gcc432.tar.gz**.

A seguir, deve-se descompactar este arquivo, digitando, num terminal: **tar zxvf ncl_ncarg-5.2.1.Linux_i686_gcc432.tar.gz**, que criará três diretórios: *bin/*, *lib/* e *include/*.

Como administrador do sistema (usuário *root*), realize os seguintes passos:

1. Criar um diretório chamado *ncarg* em */usr/local* (local padrão de instalação), digitando o seguinte comando no terminal do Linux: **mkdir /usr/local/ncarg**
2. Mover os diretórios *bin/*, *lib/* e *include/* para o diretório */usr/local/ncarg*, digitando o comando: **mv bin/ lib/ include/ /usr/local/ncarg**

Caso não seja permitida esta operação, por não ser o administrador do sistema, é possível instalar o NCL na área de trabalho do usuário, ou seja, em */home/<usuário>/ncarg* (sendo <usuário> o nome do usuário com acesso ao sistema). Suponha que o usuário *manoel* quer instalar o NCL em sua área de trabalho local. Assim, ele realizará os seguintes passos:

1. Criar o diretório *ncarg* em seu diretório de trabalho, com o comando: **mkdir /home/manoel/ncarg**
2. Mover os diretórios *bin/*, *lib/* e *include/* para o diretório */home/manoel/ncarg*, digitando o comando: **mv bin/ lib/ include/ /home/manoel/ncarg**

[1] É necessário realizar um cadastro gratuito e rápido neste site para se ter acesso ao download dos arquivos de instalação do NCL.

Agora, falta apenas configurar o ambiente do Linux para que se possa usar o NCL. Antes, é preciso saber qual é o shell usado em seu sistema. Obtenha esta informação digitando no terminal do Linux o comando **echo $SHELL**.

Se a resposta deste comando for */bin/bash*, você deve editar ou criar o arquivo *.bashrc*, localizado no diretório de trabalho do usuário, e inserir as seguintes linhas:

```
NCARG_ROOT=/usr/local/ncarg
PATH=$NCARG_ROOT/bin:$PATH
export NCARG_ROOT PATH
```

se o NCL foi instalado em seu local padrão, como administrador do sistema, ou

```
NCARG_ROOT=$HOME/ncarg
PATH=$NCARG_ROOT/bin:$PATH
export NCARG_ROOT PATH
```

se o NCL foi instalado na área de trabalho do usuário.

Se a resposta ao comando **echo $SHELL** for */bin/csh*, você deve editar ou criar o arquivo *.cshrc*, também localizado no diretório de trabalho do usuário, e inserir as seguintes linhas:

```
setenv NCARG_ROOT /usr/local/ncarg
setenv PATH $NCARG_ROOT/bin:$PATH
```

se o NCL foi instalado em seu local padrão ou

```
setenv NCARG_ROOT $HOME/ncarg
setenv PATH $NCARG_ROOT/bin:$PATH
```

se o NCL foi instalado no diretório de trabalho do usuário.

**DICA**: caso o terminal no qual as operações acima tenham sido realizadas não tenha sido fechado, deve-se carregar o arquivo *.bashrc* à memória. Para isso, digite no terminal: **source** *.bashrc*. Não esqueça do ponto antes de *bashrc*! Ele faz parte do nome do arquivo; neste caso, um arquivo oculto.

## 1.2 Executando o NCL

O NCL pode ser executado interativamente ou em modo de lote ("batch mode")[2]. O primeiro permite que os comandos sejam digitados um a um dentro do ambiente do NCL, enquanto que o segundo é usado, normalmente, para automatizar tarefas.

O NCL é sensível à caixa de texto, ou seja, *A* e *a* são considerados diferentes (ao contrário do Fortran, por exemplo). Todas as linhas de comando requerem um caractere de final de linha ("carriage return", fornecido ao pressionar ENTER) para finalizá-las.

**Modo interativo**: abaixo é apresentada uma sessão simples de uso interativo do NCL.

```
$ ncl
 Copyright (C) 1995-2010 - All Rights Reserved
 University Corporation for Atmospheric Research
 NCAR Command Language Version 5.2.1
 The use of this software is governed by a License Agreement.
 See http://www.ncl.ucar.edu/ for more details.
ncl 0> a=(/1,2,3/)
ncl 1> print(a)


Variable: a
Type: integer
Total Size: 12 bytes
            3 values
Number of Dimensions: 1
Dimensions and sizes: [3]
Coordinates:
(0) 1
(1) 2
(2) 3
ncl 2> quit
$
```

Neste exemplo, um vetor *a*, com 3 elementos numéricos, é criado e exibido na tela com o comando **print**. Aqui $ representa o *prompt* do Linux.

[2]Aqui, lote refere-se à execução de scripts, que podem ser considerados como um lote ou conjuntos de comandos do NCL

**Modo de lote** (*Batch mode*): abaixo são apresentadas três formas para se executar scripts em NCL, assumindo que o script encontra-se no arquivo *teste.ncl* e que o shell Bash está sendo usado:

```
ncl teste.ncl
ncl teste.ncl >& teste.out
ncl teste.ncl >& teste.out &
ncl < teste.ncl                        [aceitável]
```

No segundo e terceiro exemplos é especificado um arquivo *teste.out*, que receberá todo e qualquer texto que seria impresso na tela.

**DICA**: os arquivos em lote, ou scripts, têm, frequentemente, o sufixo ".ncl". Isto é apenas uma convenção, não sendo, portanto, necessário. Mesmo assim, o uso do sufixo ".ncl" é **recomendado**, pois permite a identificação dos scripts em um diretório com vários arquivos.

O NCL permite a especificação de várias opções na linha de comando. A linha de comando "ncl -h" exibirá as opções atualmente suportadas. A atribuição de variáveis em linha de comando é discutida no Capítulo 6.

As funções e/ou subrotinas e variáveis localizados em arquivos externos podem ser acessados por meio de

```
load "minhas_funcoes.ncl"
load "$HOME/meus_scripts/funcoes.ncl"
external DEMO "minha_funcao.so"
```

que podem ser digitadas dentro do ambiente interativo do NCL ou no início de um script em NCL. As duas primeiras linhas mostram como carregar variáveis e funções e/ou subrotinas localizados em outros arquivos contendo scripts em NCL. A última linha mostra como carregar uma determinada função ou subrotina (DEMO), contida na biblioteca compartilhada *minha_funcao.so*. O uso de códigos externos, feitos em C e Fortran, é abordado no Capítulo 7.

# Capítulo 2

# A Linguagem

## 2.1 Símbolos

Os símbolos de sintaxe normalmente usados incluem:

| | |
|---|---|
| ; | inicia um comentário |
| @ | cria/referencia atributos |
| ! | cria/referencia dimensões nomeadas |
| & | cria/referencia uma variável de coordenada |
| {...} | usado para subindexação de coordenada |
| $ | encerra strings ao importar/exportar variáveis via **addfile** |
| [...] | sub-indexa variáveis do tipo lista |
| (/.../) | constrói uma arranjo (vetor ou matriz) |
| : | usado na sintaxe de arranjos |
| \| | usado como um separador para dimensões nomeadas |
| \ | indica que a declaração continua na próxima linha |
| :: | usado como um separador ao chamar códigos externos |
| -> | usado para a entrada/saída de dados de arquivos em um dos formatos suportados |

Estes símbolos serão usados ao longo deste guia em exemplos práticos, portanto, o usuário não deve se preocupar com eles agora.

## 2.2 Tipos de dados

O NCL permite a manipulação de dados numéricos e não-numéricos em vários formatos distintos.

**Numéricos**: *double* (64 bits), *float* (32 bits), *long* (32 ou 64 bits), *integer* (32 bits), *short* (16 bits) e *byte* (8 bits). O tipo *complex* não é suportado.

**Não-numéricos**: *string*, *character*, *graphic*, *file*, *logical* e *list*.

## 2.3 Palavras-chave reservadas

Assim como em toda linguagem de programação, o NCL possui um conjunto de palavras-chave que não podem ser usadas para nomear variáveis e funções. Estas palavras-chave estão, normalmente, associadas a funções, comandos e tipos de dados do próprio NCL.

**begin, break, byte, character, continue, create, defaultapp, do, double, else, end, external, False, file, float, function, getvalues, graphic, if, integer, load, local, logical, long, new, noparent, numeric, procedure, quit, Quit, QUIT, record, return, setvalues, short, string, then, True, while,** e todos os nomes de funções e procedimentos internos (built-in).

## 2.4 Expressões

As regras de precedência pode ser alteradas pelo uso de parênteses "(...)" ao redor das expressões. O NCL não opera em nenhum elemento de arranjos[1] cujo valor foi definido como **_FillValue** (veja seção 2.13).

**Operadores algébricos:**

[1]Arranjos, ou array, é um termo usado em informática que refere-se a conjuntos de elementos, geralmente, do mesmo tipo de dados. Os arranjos podem ser unidimensionais (vetores), bidimensionais (matrizes) e multidimensionais.

| | |
|---|---|
| + | adição (+ é um operador sobrecarregado. Ele também é usado na concatenação de strings) |
| - | subtração |
| * | multiplicação |
| $\wedge$ | exponenciação |
| % | módulo, somente inteiros (integer) |
| # | multiplicação matricial |
| >,< | maior que, menor que |

Abaixo, exemplos de expressões algébricas que podem ser verificados com o comando **print**:

```
x=10+11
y=10-11
z=10*11.1
k=x^2
l=10%11
m=10<11
```

**Operadores lógicos:**

| | |
|---|---|
| .lt. | menor que |
| .le. | menor ou igual a |
| .gt. | maior que |
| .ge. | maior ou igual a |
| .ne. | diferente de |
| .eq. | igual a |
| .and. | e |
| .or. | ou |
| .xor. | ou exclusivo |
| .not. | não |

Abaixo, exemplos de expressões lógicas que também podem ser verificados com o comando **print**:

```
x=10.lt.11
y=10.le.11
z=10.gt.11.1
i=10.eq.11
k=.not.x
l=10.ne.11
m=x.and.y
n=x.or.y
m=x.xor.y
```

## 2.5 Variáveis

Os nomes das variáveis devem começar com um caractere alfabético, mas podem conter qualquer combinação de caracteres numéricos e alfabéticos. O uso do símbolo "_" também é permitido. As variáveis podem ter informações auxiliares (frequentemente chamadas de metadados) anexadas. Os metadados pode ser acessados, criados, modificados e apagados por meio da sintaxe e funções disponíveis no NCL (conforme mostrado nas seções 2.10-2.12)

As variáveis importadas por meio da função **addfile** conterão automaticamente todos os metadados disponíveis associados com elas.

Exemplos de nomes de variáveis válidos: *teste, teste123, teste_123, _123teste, _123*

Exemplos de nomes de variáveis inválidos: *123teste, quit, begin, @123*

A criação de variáveis pode ser feita de dois modos: *por atribuição* ou com o comando **new**. No primeiro modo, o tipo de dado associado à variável é definido pelo valor atribuído a ela. Por exemplo, *x=10* cria uma variável *x* do tipo inteiro (*integer*). Já *teste=False* cria uma variável *teste* do tipo lógico (*logical*). No caso de *lista=(/"um","dois","tres"/)*, cria-se uma variável *lista*, contendo três strings. Em outras palavras, cria-se um vetor do tipo *string*.

Certas situações demandam a criação de variáveis vazias para futuro preenchimento - normalmente para vetores e matrizes, Neste caso, o comando **new** pode facilitar a vida. Suponha que deseja-se criar uma matriz do tipo inteira com duas linhas e três colunas, que será preenchida posteriormente. Neste caso,

x = **new(** (/2,3/), *integer* **)**

A sintaxe do comando **new** é

*variável* = **new(** *tamanho das dimensões*, *tipo de dado*, *_FillValue* **)**

que, comparada ao exemplo dado, apresenta uma opção a mais, opcional: *_FillValue*. Nesta opção define-se o valor a ser usado nos elementos aos quais nenhum valor foi atribuído. Como isto não foi especificado na criação de *x*, o *_FillValue* padrão é usado, ou seja, -999. Verifique com o comando **print**.

## 2.6 Controlando a execução de comandos

Normalmente, uma tarefa é realizada por meio de uma sequência de comandos, que devem ser executados um após o outro. Entretanto, a sequência de execução de comandos pode ter de ser alterada para que a tarefa seja satisfatoriamente realizada. O NCL fornece meios de controle de execução de comandos. É possível repetir comandos por um certo número de vezes ou de acordo com uma condição, bem como escolher entre dois conjuntos de comandos se um critério for atendido ou não.

### 2.6.1 Laços de repetição

É comum a necessidade de repetição de partes de um código para a realização de uma tarefa, particularmente comum quando trata-se de arranjos multidimensionais. Em linguagem de programação dá-se a este tipo de mecanismo o nome de *laço de repetição* ou *loops*.

O uso de loops deve ser minimizado em uma linguagem interpretada. Frequentemente, os loops podem ser substituídos por uma sintaxe de arranjos ou por uma função interna. Se múltiplos loops são necessários e a velocidade de execução é importante, o uso de códigos externos, em Fortran ou em C, podem ser a melhor escolha (ver Capítulo 7).

O NCL disponibiliza dois mecanismos de repetição de comandos:

**do ... end do**

Sintaxe:

**do** *n=início,fim,incremento_ opcional*
  [*comandos*]
**end do**

sendo *n* uma variável do tipo *integer*; *início* e *fim* o intervalo de variação da variável *n*, que pode modificar-se de acordo com *incremento_ opcional.* Entre **do** e **end do** são colocados os comandos que deseja-se repetir. Veja o exemplo,

**do** *i=1,100*
  print("O valor de i eh "+i)
**end do**

Neste exemplo, a frase "O valor de i eh ", seguido do valor da variável *i* é impresso na tela 100 vezes. O exemplo seguinte mostra o uso do incremento opcional:

**do** *i=1,100,2*
  print("O valor de i eh "+i)
**end do**

que imprimirá na tela, 50 vezes, a mesma frase do exemplo anterior. Compare os valores de *i* entre estes dois exemplos.

**do while ... end do**

Sintaxe:

**do while** (*expressão_ lógica_ escalar*)
  [*comandos*]
**end do**

sendo *expressão_ lógica_ escalar* uma expressão que resulte num valor lógico, ou seja, verdadeiro (True) ou falso (False). Enquanto esta expressão for considerada verdadeira, os comandos entre **do while** e **end do** serão realizados. Veja um exemplo,

x=1
**do while** (x.le.10)

```
        print("O valor de x eh "+x)
        x=x+1
    end do
```

que mostra que os comandos entre **do while** e **end do** são executados enquanto $x$ for menor ou igual que (operador lógico *.le.*) 10.

### break e continue

Há dois comandos que são usados para alterar o funcionamento normal dos laços de repetição **do ... end do** e **do while ... end do**. O primeiro é o comando **break**, que permite interromper a execução dos comandos internos ao loop, "saltando" ao primeiro comando após o **end do**. Veja o exemplo,

```
    x=1
    do while (x.le.10)
        print("O valor de x eh "+x)
        x=x+1
        break
    end do
```

que imprimirá apenas uma vez a frase dada ao comando **print**. O segundo é o comando **continue**, que permite interromper a sequência normal de execução dos comandos internos ao loop, voltando ao primeiro comando interno ao loop, sem sair dele. Veja o exemplo,

```
    x=1
    do while (x.le.10)
        print("O valor de x eh "+x)
        x=x+1
        continue
        print("Esta frase nunca serah impressa!!)
    end do
```

que não imprimirá na tela a frase "Esta frase nunca serah impressa!!", pois o comando **continue** ordena que o loop volte ao primeiro comando, não executando o comando **print** que localiza-se após **continue**.

### 2.6.2 A construção if ... else ... end if

Por vezes, pode ser importante estabelecer um critério que definirá que conjunto de comandos será executado. Este critério é fornecido por meio de uma expressão lógica, ou seja, uma expressão que compara dois valores e/ou variáveis, resultando num valor lógico, ou seja, verdadeiro ou falso. O NCL oferece, para este fim, a construção **if ... else ... end if**.

Sintaxe(1):

```
if (expressão_lógica_escalar) then
   [comandos]
end if
```

Sintaxe(2):

```
if (expressão_lógica_escalar) then
   [comandos]
else
   [comandos]
end if
```

O exemplo abaixo mostra que se $x$ for maior que zero, os comandos entre **if** e **end if** são executados.

```
x=1
if (x .gt. 0) then
   print("x eh maior que zero!")
end if
```

O exemplo é igual ao anterior, mas inclui um comando adicional para ser executado no caso da condição verificada (*x .gt. 0*) for falsa.

```
x=1
if (x .gt. 0) then
   print("x eh maior que zero!")
else
   print("x eh menor ou igual a zero!")
end if
```

É possível combinar condições, utilizando os operadores lógicos **.and.**, **.or.** e **.xor.**. Veja um exemplo abaixo:

```
x=1
if (x .ge. 0 .and. x .le. 10) then
```

```
   print("x estah entre zero e 10!")
else
   print("x estah fora do intervalo [0,10]!")
end if
```

## 2.7 Dimensões e indexação

Há duas maneiras de se referenciar a elementos de arranjos: a padrão e por coordenadas. De forma geral, elas são similares àquelas disponíveis em outras linguagens como Fortran 90, Matlab e IDL. Em outras palavras, elas são feitas da seguinte maneira:

```
início : fim : intervalo
```

sendo *início* e *fim* os índices inicial e final que determinarão o grupo de elementos que serão acessados. A opção *intervalo*, que é opcional, altera a variação do índice, entre os limites inferior (*início*) e superior (*fim*), que por padrão é 1. Este mesmo esquema funciona para acesso via coordenadas.

### 2.7.1 Indexação padrão

No NCL, os índices iniciam em 0 e terminam em N-1, sendo N o número total de elementos (Em Fortran, por exemplo, os índices de arranjos começam em 1 e terminam em N). Considerando um arranjo unidimensional, com $N = 6$ elementos,

```
x=(/1,2,3,4,5,6/)
```

os elementos de $x$ podem ser acessados individualmente,

```
print(x(0))        ; imprime 1
print(x(4))        ; imprime 5
```

ou em grupos,

```
print(x(0:2))        ; imprime 1, 2 e 3
print(x(3:))        ; imprime 4, 5 e 6
```

Note que na construção *início:fim*, a ausência de um deles equivale ao acesso desde o primeiro ou até o último, respectivamente. Assim, consequentemente, $x(:)$ é equivalente a todos os elementos.

### 2.7.2 Indexação por coordenadas

O uso da indexação por coordenadas requer que as dimensões de um arranjo tenham um nome e que tenham variáveis de coordenada atribuídas a elas.Maiores detalhes são dados na seção 2.10. A indexação por coordenadas é usada colocando-se as coordenadas entre chaves "...". Para fins de exemplificação, as linhas de comando do NCL abaixo mostram a utilização desta forma de indexação:

```
x=(/1,2,3,4,5,6/)
x!0="lat"
x&lat=(/-10,-20,-30,-40,-50,-60/)
print(x(1))
print(x({-20}))
```

Note que os dois comandos **print** imprimem o mesmo resultado, ou seja, acessam o mesmo elemento do vetor *x*. O primeiro usa a indexação padrão, enquanto o segundo usa as informações de coordenadas atribuídas aos dados. Fica óbvia a facilidade de acesso a certos elementos de um arranjo ao se utilizar coordenadas.

## 2.8 Redução de dimensão

Quando uma constante é usada na indexação de um arranjo - padrão ou por coordenadas - ocorre uma redução de dimensão. Isto é ilustrado pelas linhas de comando abaixo:

```
y = (/ (/10,20,30,40/),(/20,30,40,50/) /)
y1 = y(0,:)
print(y1)
```

Na primeira linha um arranjo bidimensional contendo duas linhas e quatro colunas é armazenado na variável *y*. Na segunda linha usa-se uma constante na primeira dimensão de *y*. A variável *y1* contem uma dimensão a menos que *y*, ou seja, foi provocada uma diminuição da dimensão de *y*; ela contém

apenas a primeira linha do arranjo $y$. Esta operação não provoca perda de metadados, exceto aqueles relacionados à dimensão eliminada. Caso não se queira perder estes metadados, pode-se forçar a retenção da dimensão eliminada. Seguindo o exemplo anterior,

```
y2 = y(0:0,:)
```

Esta linha de código provoca a retenção da primeira dimensão de $y$. Aplique a função **printVarSummary** às variáveis *y1* e y2 para verificar as diferenças.

## 2.9 Dimensões nomeadas

Para ilustrar o uso de coordenadas na indexação de arranjos foi necessário dar um nome a dimensão do vetor usado no exemplo. Ou seja, precisou-se nomear a dimensão do vetor (neste caso, obviamente, única). O símbolo "!" é usado para se atribuir um nome a (ou obtê-lo de) uma dimensão de um arranjo. A numeração das dimensões iniciam da esquerda para direita, com a dimensão mais a esquerda igual a 0. Do exemplo da seção anterior, vamos dar nomes às dimensões do arranjo bidimensional $y$:

```
y!0 = "lat"
y!1 = "lon"
```

Note que o nome é uma string de caracteres e, portanto, deve ser informado entre aspas duplas.

### 2.9.1 Reordenação de dimensões

Uma vez nomeadas as dimensões de um arranjo, é possível mudar as suas posições, ou seja, reordená-las. Continuando no exemplo anterior, vamos alterar a ordem das dimensões do arranjo bidimensional $y$:

```
y_reorden = y(lon|:,lat|:)
```

Agora, temos uma nova variável *y_reorden* com as mesmas dimensões de $y$, mas com a sua ordem trocada. Isso é particularmente útil em duas situações distintas: (i) operações com arranjos contendo as mesmas dimensões - tamanho e quantidade - mas cada um deles em uma ordem diferente e (ii) aplicações de funções do NCL a uma certa dimensão.

## 2.10 Variáveis de coordenadas

Pela definição netCDF, uma variável de coordenada é um arranjo unidimensional contendo valores crescentes ou decrescentes monotonicamente que tem o mesmo nome e o mesmo tamanho da dimensão a qual ela é atribuída. As variáveis de coordenadas representam as coordenadas dos dados para cada índice em uma dimensão nomeada. Elas podem ser usadas na indexação por coordenadas. O operador "&" é usado para referenciar e atribuir variáveis de coordenada. Para atribuir uma variável de coordenada a uma dimensão, a dimensão deve, em primeiro lugar, possuir um nome. Usando o exemplo anterior, vamos atribuir coordenadas ao arranjo bidimensional $y$, cujas coordenadas já possuem nomes:

```
lat = (/-10,-20/)
lon = (/-60,-50,-40,-30/)
y&lat = lat
y&lon = lon
```

Veja o resultado com a função **printVarSummary**.

## 2.11 Atributos

Atributos são informações descritivas que podem estar associadas a uma variável. Eles são muito úteis para comunicar informações ao usuário a respeito de dados específicos. Juntamente com as coordenadas, formam os metadados de uma variável. Os atributos de variáveis são atribuídos e referenciados pelo nome da variável, seguido do símbolo "@" e do nome do atributo. Suponha que o arranjo bidimensional $y$ do exemplo anterior - que já possui coordenadas - contenha dados sobre a umidade relativa de uma certa região. Vamos passar dois atributos a este arranjo:

```
y@unidade = "porcentagem (%)"
y@data = "21/11/2011"
```

Passamos atributos que informam a unidade física dos dados e a data de coleta.

## 2.12 _FillValue

O atributo **_FillValue** é uma atributo reservado do netCDF e do NCL que indica valores ausentes. Algumas operações algébricas e gráficas tratam o **_FillValue** de maneira especial. Seguindo o exemplo anterior, vamos passar este atributo ao arranjo bidimensional $y$:

```
y@_FillValue = -999
```

Qualquer elemento de $y$ contendo o valor *-999* será interpretado como dado ausente e não será usado em operações matemáticas. Veja mais sobre valores ausentes no NCL no Apêndice D.

## 2.13 Coerção

Chama-se coerção uma conversão implícita de dados de um tipo para outro. Isto ocorre quando dois valores de diferentes tipos são operandos de um mesmo operador. Um exemplo simples é:

```
z = 5.2 + 9
```

Na qual *5.2* é do tipo *float*, enquanto *9* é do tipo *integer*. Neste caso, *9* é silenciosamente convertido ao tipo *float* antes da adição. O NCL converterá automaticamente quando nenhuma informação é perdida. Neste exemplo, $z$ será uma variável do tipo *float*. Entretanto, se $k$ é do tipo *integer* e $x$ é do tipo *float* (ou do tipo *double*), então a seguinte declaração resulta num erro fatal (nenhuma conversão é possível porque há uma possível perda de informação):

```
k = x
```

Quando informação pode ser perdida, funções de conversão explícita devem ser usadas:

k = **floattointeger**(x)

## 2.14 Variáveis e metadados

Há dois tipos de atribuições em NCL: (i) de valores e (ii) variável a variável. Somente a última copia os metadados. As atribuições apenas de valores ocorrem quando o lado direito de uma atribuição não é uma variável. O lado direito pode ser uma constante, o resultado de uma expressão ou o resultado do construtor de arranjo **(/.../)**. Nenhum nome de dimensões, variáveis de coordenadas ou atributos são passados à nova variável, exceto o atributo **_FillValue**. Por meio do arranjo bidimensional *y*, usado até aqui, podemos testar os dois tipos de atribuição:

```
a = y
b = (/y/)
```

A variável *a* será uma cópia exata de *y*, com seus dados, coordenadas e atributos. Ao contrário, *b* conterá apenas os dados de *y* e o atributo _FillValue, não carregando quaisquer atributos e coordenadas. Verifique isto aplicando a função **printVarSummary** às variáveis *a* e *b*.

## 2.15 Apagando variáveis, atributos e coordenadas

A subrotina **delete** é usada para eliminar variáveis, atributos e variáveis de coordenada. Não é possível eliminar variáveis de arquivos, atributos de arquivos e coordenadas de arquivos. Esta subrotina é especialmente útil quando se deseja reutilizar uma variável, mas os novos valores e/ou tipos são diferentes da variável original.

Quando uma variável é passada à **delete**, todos os seus valores são removidos da memória. Se um atributo ou uma variável de coordenada é passada à **delete**, ele(a) é removido(a) da variável a qual pertence. Os exemplos abaixo ilustram o seu uso:

```
y = 1
y@teste = "atributo"
x = 2
list_vars()
```

```
integer y [ 1 ]
```

```
      teste

  integer x [ 1 ]

    delete( y@teste )
    list_vars()

  integer y [ 1 ]

  integer x [ 1 ]

    delete( x )
    delete( y )
```

Neste exemplo, criamos duas variáveis escalares $x$ e $y$, tendo esta última um atributo *teste*. Usamos aqui outra subrotina, para nos auxiliar a observação do funcionamento de **delete**, a subrotina **list_vars**, que lista as variáveis existentes. Note que na sua primeira execução, temos a listagem das duas variáveis criadas e do atributo de $y$.

Na primeira execução de **delete**, apagamos o atributo de $y$, como podemos ver após a segunda execução de **list_vars**. Por fim, executamos duas vezes **delete**, eliminando as variáveis $x$ e $y$.

É possível, também, eliminar mais de uma variável ao mesmo tempo. Esta capacidade está disponível a partir da versão 6.0.0-beta do NCL. Veja o exemplo abaixo:

```
y = 1
x = 2
delete( [/ x, y /] )
```

Note que foi usado uma sintaxe especial, definida por [/ ... /]. Dentro destes limitadores, passa-se uma lista de nomes de variáveis - separados por vírgulas - que deseja-se apagar.

# Capítulo 3

# Arquivos de Entrada/Saída do NCL

## 3.1 Arquivos Suportados

Os formatos de arquivo conhecidos pela linguagem NCL são chamados de **formatos suportados**. São eles: netCDF, HDF, GRIB e CCM History Tape (somente cray[1]). O site `http://www.ncl.ucar.edu/Applications/list_io.shtml` apresenta algumas informações adicionais sobre esses formatos de arquivo. Para importar dados dos arquivos suportados utiliza-se a função **addfile**, que possui a seguinte sintaxe:

*variável* = **addfile**(*caminho e nome do arquivo*, *status*)

O caminho do arquivo pode ser inteiro ou relativo à pasta atual. Os arquivos em formatos suportados tem diferentes extensões. Segue abaixo uma lista dos tipos de arquivos suportados e suas respectivas extensões:

| | |
|---|---|
| netCDF | .nc ou .cdf |
| HDF4 | .hd ou .hdf |
| HDF4-EOS | hdfeos |
| GRIB-1 ou GRIB-2 | .grb ou .grib |
| CCM History Tape (somente cray) | .ccm |

[1] É o formato de arquivo dos modelos climáticos globais CCM1, CCM2 e CCM3, escritos pelo NCAR.

O arquivo de dados não necessita de uma extensão junto ao seu nome. Por exemplo, se existe um arquivo chamado "modelo" em formato grib, usa-se "modelo.grb" dentro da função **addfile** e o NCL irá procurar primeiro por um arquivo chamado "modelo.grb" e em seguida por um arquivo chamado "modelo", tratando ele como arquivo em formato grib.

O status do arquivo pode ser "r" para somente leitura, isto é, o arquivo não pode ser editado, "c" para criação de um novo arquivo e "w" para leitura e edição, ou seja, o arquivo pode ser editado no ncl. Nem todos os formatos de arquivos suportados podem ser criados ou editados:

| | |
|---|---|
| "r" | leitura - todos os formatos suportados |
| "c" | criação - somente netCDF e HDF4 |
| "w" | leitura e edição - somente netCDF e HDF4 |

Supondo que exista o arquivo "analise.nc" (arquivo em formato netCDF), usado para a visualização de mapas, dentro da pasta "dados". Pode-se importar este arquivo da seguinte forma:

```
abrir = addfile("dados/analise.nc","r")
```

Igualmente, caso a intenção seja criar um novo arquivo chamado "analise" no formato netCDF, na pasta dados, pode-se fazer da seguinte forma:

```
abrir = addfile("dados/analise.nc","c")
```

A partir daí, a referência ao arquivo é sempre dada pela palavra "abrir", que aponta para a abertura do arquivo. Para importar uma variável dentro de um arquivo de qualquer um dos formatos suportados, usam-se os caracteres ->. Primeiramente, é necessário conhecer o nome da variável dentro deste arquivo. Por exemplo, a variável temperatura em superfície pode ter o nome *tmp* ou *tmpsup* ou *tmp2m*, conforme escrito pelo criador do arquivo. Considerando que dentro do arquivo referenciado pela palavra "abrir", a temperatura em superfície é chamada de *tmp2m*, pode-se importar os dados desta variável da seguinte forma:

```
temperatura = abrir->tmp2m
```

Assim, a palavra "temperatura" é a referência à variável *tmp2m* dentro do arquivo "dados/analise.nc". Caso o nome da variável seja escrito com algum erro, a palavra "temperatura" não encontra a variável dentro do arquivo e não recebe valor algum.

**Quando usar o *$*:**

Em alguns casos, o nome da variável deve estar rodeado pelo símbolo *$*. Isso é necessário se a variável no arquivo tem um caracter não-alfanumérico (por exemplo + ou - dentro dele) ou então se o ítem do lado direito do apontador **->** é ele próprio uma variável do tipo *string*. Por exemplo, se um arquivo possui a variável "tmp+2m-obs" com informações de temperatura em 2 metros, usa-se o formato com o símbolo *$* (devido aos sinais "+" e "-"):

```
temperatura = abrir->$tmp+2m-obs$
```

No segundo exemplo, as letras "T", "U" e "V" formam uma variável do tipo *string*, portanto também usa-se o formato com o símbolo *$*:

```
vars = (/"T","U","V"/)
variaveis = abrir->$vars(n)$ ;n=0,1,2,3
```

**Imprimindo o conteúdo de um arquivo em formato suportado:**

A função **print** pode ser usada para visualizar o conteúdo de qualquer arquivo em formato suportado. A informação visualizada é similar àquela mostrada pelo aplicativo netCDF/Unidata utilizando o comando "ncdump -h *nome-doarquivo.nc*". Então, tendo o arquivo "gfs00.grib2", pode-se visualizar as informações que estão contidas nele com o comando:

```
abrir = addfile("gfs00.grib2","r")
print(abrir)
```

Informações como nome do arquivo, dimensões (graus de latitude e longitude, níveis verticais...), variáveis (tipo, nome, unidade, centro de origem, nível...) são mostradas na tela. Essa é uma ferramenta útil para se identificar quais as variáveis e seus nomes dentro do arquivo.

**Reordenando uma variável na entrada:**

Assumindo que *TMP* é uma variável 3D de tamanho (ntempo,nlat,nlon). Para inverter a latitude da matriz:

```
temperatura = abrir -> TMP (:,::-1,:)
```

**Importando dado do tipo byte ou short:**

Um grupo de funções, escritas por usuários do NCL, é distribuída junto com a instalação do programa. Várias funções nesta biblioteca convertem variáveis do tipo *short* e *byte* para *float*. Por exemplo:

```
temperatura = short2flt(abrir->TMP)
temperatura = byte2flt(abrir->TMP)
```

No primeiro exemplo a variável *TMP* é originalmente do tipo *short*, porém temperatura é do tipo *float*. Semelhantemente, no segundo exemplo a variável *TMP* é originalmente do tipo *byte*, mas temperatura é do tipo *float*. Antes de usar essas funções, é necessário carregar a biblioteca das contribuições, colocando a linha abaixo:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
```

### 3.1.1 Abrangendo múltiplos arquivos:

A função **addfiles** permite o acesso de dados de múltiplos arquivos. Essa função pode concatenar registros e também agrupar variáveis de diferentes arquivos adicionando uma dimensão extra. Pode-se agrupar, por exemplo, arquivos de uma mesma rodada e horários de previsão diferentes. Neste caso, é útil usar a função **systemfunc** em conjunto, para listar todos os arquivos de dados antes de abri-los. Supondo que os arquivos de uma determinada rodada do modelo gfs tenham os nomes: "gfsprevisao00.grb", "gfsprevisao06.grb", "gfsprevisao12.grb" e assim por diante. Para abri-los em conjunto e extrair a variável temperatura em 2 metros (*TMP2M*), pode-se usar os seguintes comandos:

```
arquivos = systemfunc("ls gfsprevisao*.grb")
abrir = addfiles(arquivos,"r")
temperatura = abrir[:]->TMP2M
```

Neste exemplo, a variável resultante "temperatura" agrupa as previsões de todos os horários do modelo gfs, mesmo que originalmente estejam em arquivos diferentes.

O procedimento **setfileoption** pode ser usado para alterar aspectos do comportamento das funções **addfile** e **addfiles**. Para mais informações consulte a página de documentação do ncl: `http://www.ncl.ucar.edu/Document/Functions/Built-in/setfileoptions.shtml`.

## 3.2 Arquivos de Dados Binários

Dados binários não são necessariamente portáteis. A maior parte das máquinas escrevem binários IEEE. Uma exceção notável é o binário nativo CRAY. Até mesmo binários IEEE não são necessariamente portáteis. Os binários IEEE são de duas formas diferentes: *big endian* e *little endian*. Dependendo da máquina, um arquivo binário pode ter que reordenar os *bytes* (*byte swap*). O NCL permite reordenamento de *bytes* dinâmico via procedimento **setfileoption**.

Pode ser útil abrir ou escrever arquivo binário no formato Fortran sequencial não formatado, que é um arquivo semelhante àquele criado no Fortran com FORMAT=UNFORMATTED e ACCESS=SEQUENTIAL(padrão). Arquivos com acesso *sequential* contém cabeçalho com informações de tamanho de registro, já arquivos com acesso *direct* não contém informações de tamanho de registro em seu cabeçalho.

### 3.2.1 Lendo arquivos binários:

Várias funções lêem dados de arquivos binários. A maior parte lê binários IEEE e vários lêem binários CRAY.

A função **fbinrecread** pode ser usada para leitura de um arquivo Fortran sequencial não-formatado. Os registros, que começam com 0, podem ser de vários tipos e tamanhos.

Sintaxe:
*variável* = **fbinrecread**(*caminho do arquivo*, *número do registro*, *dimensões*, *tipo*)

Por exemplo, para ler o registro 0 de um arquivo com o nome "arquivo.ieee":

```
abrir = fbinrecread("arquivo.ieee", 0, (/64,128/), "float")
```

A função **fbindirread** pode ser usada para leitura de registros binários de tamanho de registro fixo (acesso direto). Todos os registros do arquivo devem ser do mesmo tipo e dimensão.

Sintaxe:
*variável* = **fbindirread**(*caminho do arquivo*, *número do registro*, *dimensões*, *tipo*)

Por exemplo, para ler o registro (n+1) do arquivo "arquivo.ieee":

```
abrir = fbindirread("arquivo.ieee",n,(/73,144/),"float")
```

Outras funções para a leitura de dados binários incluem: **cbinread**, **fbinread** e **craybinrecread**.

### 3.2.2 Escrevendo arquivos binários:

Vários procedimentos escrevem arquivos IEEE de dados binários.

A função **fbinrecwrite** pode ser usada para escrever um arquivo Fortran sequencial não formatado.

Sintaxe:
*variável* = **fbinrecwrite**(*caminho do arquivo*, *número do registro*, *valor*)

Os registros podem ser de vários tipos e tamanhos. Por exemplo, assumindo que existem as cinco variáveis *tempo(ntime)*, *lat(nlat)*, *lon(nlon)*, *y(ntime, nlat, nlon)* e *z(nlat, nlon)* (note que usando -1 como número de registro significa adicionar):

```
fbinrecwrite("nomedoarquivo.ieee",-1,tempo)
fbinrecwrite("nomedoarquivo.ieee",-1,lat)
fbinrecwrite("nomedoarquivo.ieee",-1,lon)
fbinrecwrite("nomedoarquivo.ieee",-1,y)
fbinrecwrite("nomedoarquivo.ieee",-1,z)
```

A função **fbindirwrite** pode ser usada para escrever um arquivo Fortran de acesso direto (ACCESS=DIRECT) não formatado. Todos os registros devem ser do mesmo tamanho e tipo. Se o arquivo já existir, o valor será adicionado àqueles já existentes no arquivo.

Sintaxe:
*variável* = **fbindirwrite**(*caminho do arquivo*, *valor*)

Por exemplo, para criar um arquivo Fortran de acesso direto não formatado com 100 registros e nome samp.dat:

```
nmaximo=100
do x=0,nmaximo-1
```

```
    fbindirwrite("/meu/diretorio/samp.dat",x)
  end do
```

**DICA**: Esse é semelhante ao exemplo 3 do guia do usuário do Grads (`http://www.iges.org/grads/gadoc/aboutgriddeddata.html#create`) para a criação de um arquivo Fortran de acesso direto não formatado com 100 registros. Verifique o site e compare as duas formas de criação deste tipo de arquivo.

Outros procedimentos para a leitura de dados binários IEEE incluem: **cbinwrite** e **fbinwrite**.

**Lendo e escrevendo arquivos *big endian* e *little endian*:**

No NCL, o comportamento padrão das funções de leitura/escrita dos binários e procedimentos são de assumir que os arquivos são compatíveis com o tipo *endian* para o sistema. O procedimento **setfileoption** pode ser usado para executar dinamicamente *byte swapping*, se necessário.

## 3.3 Arquivos de Dados ASCII

É muito comum dados meteorológicos, especialmente aqueles advindos de estações meteorológicas e radiossondagens, estarem disponíveis em arquivos ASCII. Arquivos ASCII são arquivos em texto puro, sem qualquer formatação e abertos por editores de texto simples como o *Bloco de notas*, no Windows, e *gedit*, no Linux, por exemplo. Como estes arquivos não seguem um padrão no modo em que os dados são organizados, como ocorre com os arquivos em formato suportado diretamente pelo NCL (netCDF, HDF, GRIB ou CCM), os arquivos ASCII não podem ser abertos com a função **addfile**. Entretanto, o NCL dispõe de funções de leitura e escrita de arquivos ASCII, mas deixa sob a responsabilidade do usuário a organização dos dados em uma maneira que possibilite a sua utilização.

### 3.3.1 Abrindo arquivos ASCII

Os arquivos em formato ASCII (arquivo texto) são abertos com a função **asciiread**, cuja sintaxe básica é

*variável* = **asciiread(***caminho do arquivo*, *dimensão*, *tipo***)**

sendo "dimensão" que indica o número de linhas e colunas dos dados. Pode-se passar o valor "-1" como dimensão dos dados. Neste caso, a variável que conterá os dados será um arranjo unidimensional (vetor), cujo tamanho é o número total de dados contidos no arquivo ASCII. Já o "tipo" especifica que tipo de dado será lido, ou seja, se é *string*, *float* ou *integer*.

Para ilustrar a abertura de arquivos ASCII vamos supor que há um arquivo chamado "dados.txt", contendo uma matriz com 3 linhas e 3 colunas, como mostrado abaixo:

```
12  9  3
24 12  6
28 14  7
```

Podemos carregar os dados deste arquivo com a seguinte linha de comando:

```
matriz = asciiread( "dados.txt", (/3,3/), "integer")
```

Esta linha criará um arranjo bidimensional do tipo inteiro contendo 3 linhas e 3 colunas com o nome *matriz*. Podemos ler o mesmo arquivo usando "-1" para a dimensão, ficando como:

```
dados = asciiread( "dados.txt", -1, "integer")
```

Agora, como não passamos nenhuma informação a respeito do número de linhas e colunas, os dados serão lidos e atribuídos em um arranjo unidimensional. Neste exemplo, a variável *dados* será um vetor contendo 9 valores inteiros.

### 3.3.2 Escrevendo arquivos ASCII:

A subrotina **asciiwrite** cria um arquivo ASCII com o conteúdo de uma variável. Independente do número e do tamanho das dimensões da variável passada à subrotina **asciiwrite**, ela criará um arquivo ASCII contendo um vetor de informações, ou seja, inúmeras linhas de dados contendo apenas uma

coluna de valores. Em outras palavras, não é possível controlar a formatação do arquivo ASCII ao se utilizar esta subrotina. Imagine que queremos escrever o conteúdo do arranjo bidimensional *matriz* apresentado acima, num arquivo chamado "copia_dados.txt". Para tal, usa-se a seguinte linha de comando:

```
asciiwrite("copia_dados.txt",matriz)
```

O arquivo "copia_dados.txt" conterá uma única coluna de dados:

```
12
9
3
24
12
6
28
14
7
```

Já a subrotina **write_matrix** pode escrever múltiplas colunas em um formato que pode ser controlado. Dessa forma, pode-se escrever valores em várias colunas. Vamos escrever o conteúdo da variável *matriz* num arquivo chamado "copia2_dados.txt":

```
opcoes = True
opcoes@fout = "copia2_dados.txt"
write_matrix( matriz, "3i5", opcoes )
```

Neste caso, o arquivo "copia2_dados.txt" será uma cópia exata do arquivo "dados.txt" definido no início desta seção. Note que o nome do arquivo é passado à subrotina **write_matrix** por um atributo, *fout*, que é passado à variável *opcoes*. Caso esta variável possuísse um valor igual a *False*, a subrotina imprimiria a matriz na tela.

A formatação do arquivo é definida no segundo argumento da subrotina **write_matrix**, que é definida como "3i5". Esta formatação segue a notação do Fortran. O leitor é convidado a visitar o site oficial do NCL para conhecer as outras formas de formatação e/ou consultar bibliografias sobre Fortran. Neste exemplo, o primeiro caracter da formatação define o número de colunas - 3 colunas - o segundo caracter define o tipo de dado - inteiro (integer) - e o terceiro o tamanho da cada coluna de dada - 5 dígitos. Assim, cada linha do arquivo "copia2_dados.txt" terá a forma:

aaaaabbbbbccccc

sendo *aaaaa* o primeiro conjunto (coluna) de 5 dígitos, *bbbbb* o segundo conjunto (coluna) de 5 dígitos e *ccccc* o último conjunto de 5 dígitos, formando os três conjuntos de valores inteiros.

## 3.4 Escrevendo arquivos netCDF/HDF

Os arquivos escritos em formato netCDF são mais portáteis que arquivos do tipo ASCII, ou seja, conseguem armazenar maior quantidade de dados em menos espaço de máquina. Por isso, é válido criar arquivos em formato netCDF para guardar grandes quantidades de dados meteorológicos. O NCL tem duas abordagens de criação de arquivos netCDF (ou HDF). O primeiro método é chamado de **método simples** enquanto que o segundo segue o **método tradicional** de predefinição explícita do conteúdo do arquivo antes de escrever qualquer valor dentro dele.

**Método simples:**

Esse método é direto (pode-se substituir ".hdf" por ".nc" para criar um arquivo HDF). Por exemplo:

```
abrir = addfile("nomedoarquivo.nc","c")
abrir->TMP2M = temperatura
abrir->UWIND = ventozonal
```

Para criar uma dimensão ilimitada, que normalmente é o tempo, é necessário adicionar a linha de código abaixo antes de escrever qualquer valor:

```
filedimdef(abrir,"time",-1,True)
```

**Método Tradicional**

Em alguns casos, o **método simples** de criação de arquivo netCDF é muito lento, particularmente se existem muitas variáveis ou a saída resultante do arquivo é muito extensa. O **método tradicional** é mais eficiente. Esse método requer que o usuário defina explicitamente o conteúdo do arquivo de entrada, antes de escrevê-lo. As funções do NCL que predefinem um arquivo netCDF:

**filevardef:** define o nome de uma ou mais variáveis;

**filevarattdef:** copia os atributos de uma variável para uma ou mais variáveis;

**filedimdef:** define dimensões, inclusive dimensões ilimitadas;

**fileattdev:** copia os atributos de uma variável para um arquivo com atributos globais;

**setfileoption:** algumas opções podem melhorar dramaticamente o desempenho da criação do arquivo.

Para exemplificar, assume-se que as variáveis *time*, *lat*, *lon* e *T* estão em memória. Para escrever um arquivo chamado "saida.nc", em que a variável *T* recebe o nome *TMP*:

```
abrir = addfile("saida.nc","c")
;Cria os atributos globais:
fileAtt = True
fileAtt@title = "Exemplo de arquivo"
fileAtt@Conventions = "None"
fileAtt@creation_date = systemfunc("date")
fileAttdef(abrir,fileAtt)

;Predefine as coordenadas das variáveis:
;-1 significa que a dimensão é ilimitada (pode crescer)
dimNames = (/"time","lat","lon"/)
dimSizes = (/-1,nlat,nlon/)
dimUnlim = (/True,False,False/)

;Predefine nomes, tipos e dimensões:
filedimdef(abrir,dimNames,dimSizes,dimUnlim)
filevardef(abrir,"time",typeof(time),getvardims(time))
filevardef(abrir,"lat",typeof(lat),"lat")
filevardef(abrir,"lon","float","lon")
filevardef(abrir,"TMP",typeof(T),getvardims(T))
;Predefine cada atributo de variável:
filevarattdef(abrir,"time",time)
filevarattdef(abrir,"lat",lat)
filevarattdef(abrir,"lon",lon)
filevarattdef(abrir,"TMP",T)

;Escreve os valores somente:
;trocando o nome da variável de T para TMP
abrir->time = (/time/)
```

```
abrir->lat = (/lat/)
abrir->lon = (/lon/)
abrir->TMP = (/T/)
```

### 3.4.1 Escrevendo escalares em netCDF:

Tanto no método simples como no método tradicional, o NCL reserva o nome **"ncl_scalar"** para identificar valores escalares que devem ser escritos em um arquivo netCDF.

No método simples:

```
abrir = addfile("nome_simples.nc","c")
con = 5
con!0 = "ncl_scalar"
abrir->constant = con
```

No método tradicional:

```
re = 6.37122e06
re@long_name = "radius of earth"
re@units = "m"
abrir = addfile("nome_tradicional.nc","c")
filevardef(abrir,"re",typeof(re),"ncl_scalar")
filevarattdef(abrir,"re",re)
abrir->re = (/re/)
```

**Conteúdo de um arquivo netCDF/HDF bem escrito:**

Os atributos globais do arquivo, como título, convenção e fonte, devem ser incluídos em todos os arquivos. Outros atributos adicionais podem ser incluídos como história e referências.

**Convenção de linha de comando dos formatos suportados para netCDF:**

O NCL tem um utilitário de linha de comando chamado **ncl_convert2nc**, que converte qualquer arquivo em formato suportado (GRIB-1, GRIB-2, HDF4, HDF4-EOS) para netCDF. Mais detalhes na página de documentação: `http://www.ncl.ucar.edu/Document/Tools/ncl_convert2nc.shtml`.

## 3.5 Acesso à arquivos remotos: OPenDAP

Alguns servidores de dados permitem acesso remoto de dados via **OPenDAP** (Open Source Project for Network Data Access Protocol). O acesso via **OPeNDAP** pelo NCL está disponível em alguns sistemas operacionais, via uma URL que está rodando um servidor **OPeNDAP**:

```
abrir = addfile ("http://pasta/nome.nc","r")
varios = "http://pasta/"+(/"nome1.nc","nome2.nc","nome3.nc"/)
abrirvarios = addfiles(varios,"r")
```

O usuário pode testar a disponibilidade do arquivo usando **isfilepresent**. Note que o usuário pode ser bloqueado por um firewall, não tendo acesso aos dados remotos. Além disso, alguns servidores **OPeNDAP** requerem registro, com usuário e senha cadastrado, antes de liberar o acesso aos dados.

# Capítulo 4

# Impressão

O NCL fornece capacidade de impressão limitada, em alguns casos, se obtém melhor formatação com o Fortran ou C. Funções de impressão são úteis para exibir conteúdos de variáveis e suas características e na tarefa de depuração de um script, ou seja, na procura de falhas e mal funcionamento.

As funções e as subrotinas de impressão incluem:

| | |
|---|---|
| **printVarSummary** | Fornece um resumo de uma variável, incluindo todos os metadados. |
| **printFileVarSummary** | Fornece um resumo de uma variável associada a um arquivo (presente a partir da versão 6.0.0 do NCL). |
| **print** | Fornece as mesmas informações que a função **printVarSummary**, seguidas pelos valores para cada elemento. |
| **sprinti,sprintf** | Oferece controle (limitado) sobre formato. |
| **write_matrix** | Imprime os dados na forma tabular. |

## 4.1 printVarSummary

Esta subrotina exibe as informações associadas com uma variável. Estas informações incluem tipo, atributos, tamanho de dimensões e seus nomes (se existentes) e um resumo dos dados de coordenadas (se houver). É frequentemente usado quando se deseja alguma informação da variável sem a necessidade de impressão de todos os dados. O exemplo abaixo ilustra o uso desta subrotina:

```
x=10
printVarSummary(x)
y=(/1,2,3,4,5/)
printVarSummary(y)
z=(/ (/1,2/), (/3,4/) /)
printVarSummary(z)
```

A primeira execução desta subrotina imprimirá na tela as informações a respeito da variável escalar *x*, a segunda sobre o arranjo unidimensional (vetor) *y* e a terceira imprimirá informações sobre o arranjo bidimensional *z*. As linhas abaixo mostram um resultado típico desta subrotina, neste caso, para a terceira execução apresentada acima:

```
Variable: z
Type: integer
Total size: 16 bytes
            4 values
Number of Dimensions: 2
Dimensions and sizes: [2] x [2]
Coordinates:
```

## 4.2 print

Esta subrotina exibe as informações de uma variável e o seu conteúdo - quando o seu argumento é uma variável - ou simplesmente exibe os seus valores - quando o seu argumento é o resultado de uma expressão ou um valor literal. Para variáveis do tipo *byte*, a subrotina **print** exibirá os seus valores no sistema octal. Ela aceita a indexação padrão e por coordenadas para se referir a partes específicas de um arranjo. Nenhum controle de formato está disponível.

Por definição, esta subrotina imprime os valores de uma variável precedidos pelo seu índice na variável. Escalares são precedidos por “(0)”.

Quando executado nas linhas de comando do NCL (em modo interativo) e o conteúdo a ser exibido é mais extenso que o permitido pela tela, a subrotina permite avançar manualmente (como o comando *more* do Linux).

Os exemplos abaixos mostram resultados típicos gerados a partir desta subrotina;

**Exemplo 1:**

```
p=ispan(1,4,1)
print(p)

Variable: p
Type: integer
Total Size: 16 bytes
4 values
Number of Dimensions: 1
Dimensions and Sizes: [4]
Coordinates:
(0) 1
(1) 2
(2) 3
(3) 4
```

**Exemplo 2:**

```
u=(/-53.8125,-25.4589,67.3408,98.2367,75.4578/)
print("min(u)="+min(u)+"max(u)="+max(u))

(0) min(u)=-53.8125 max(u)=98.2367
```

## 4.3 sprintf e sprinti

Estas funções usam uma string de caracteres que definem uma formatação para os dados a serem impressos. É similar a versão da linguagem C, mas possui duas diferenças importantes: (1) somente o operador % é permitido neste string de caracteres e (2) somente números inteiros (**sprinti**) e em ponto flutuante (**sprintf**) são permitidos. As linhas abaixo ilustram o uso destas funções:

**Exemplo 1:**

```
x = 23456789
print( sprintf("%6.4f", x) )
```

```
(0) 23456788.0000
print( sprintf("%6.4e", x) )
(0) 2.3457e+07
print( sprintf("%6.4E", x) )
(0) 2.3457E+07
print( sprintf("%6.4g", x) )
(0) 2.346e+07
print( sprintf("%6.4G", x) )
(0) 2.346E+07
```

**Exemplo 2:**

```
u=(/-53.8125,-62.9860,45.6590,55.2759/)
print("min(u)="+sprintf("%5.2f",min(u)))
(0) min(u) = -62.98
```

**Exemplo 3:**

```
ii=(/-47,3579,24680/)
print(sprinti("%+7.5i",ii))
(0) -00047
(1) +03579
(2) +24680
```

Abaixo é explicado brevemente como construir uma string de caracteres de formatação para a função **sprintf**:

1. Cada especificação de conversão (string de formatação) começa com um % e termina com um caractere de conversão: f, e/E, g/G.
2. Entre o % e o caractere de conversão, pode haver, na seguinte ordem:
   - Um sinal de menos, que especifica um ajuste à esquerda do argumento (valor) convertido.
   - Um número que determina a largura mínima do campo (resultado da função). O argumento convertido será impresso em um campo com esta largura, no mínimo, podendo ser preenchido se necessário.
   - Um ponto, que separa a largura do campo de sua precisão.
   - Um número que determina a precisão, ou seja, o número de dígitos após a vírgula de um valor em ponto flutuante.

3. Os caracteres de conversão são:

   - f: *[-]m.dddddd*, sendo o número de d's dado pela precisão (padrão: 6)
   - e[,E]: *[-]m.dddddd*e±xx ou *[-]m.dddddd*E±xx, sendo o número de d's dado pela precisão (padrão: 6)
   - g[,G]: use %e ou %E se o expoente é menor que -4 ou maior ou igual à precisão. Ao contrário, use %f.

Os exemplos acima apresentados ilustram a aplicação destas regras.

## 4.4 write_matrix

A subrotina **write_matrix** fornece a habilidade de imprimir tabelas a partir de arranjos bidimensionais que podem ser direcionadas para a tela ou para um arquivo. Não é possível imprimir colunas e linhas individuais de texto, que sirvam como cabeçalhos à tabela impressa. A formatação da tabela é feita por meio de *descritores de edição* do Fortran. Procure por textos sobre o Fortran para descrições mais completas. Abaixo, uma breve explicação é dada, sendo *w* referente ao tamanho total de cada valor, que deve levar em conta o sinal de menos:

- Para valores inteiros é possível usar *Iw/iw* ou *Iw.m/iw.m*.
- Para valores em ponto flutuante, os seguintes descritores de edição podem ser usados: *Fw.d/fw.d*, *Ew.d/ew.d*, *Dw.d* e *Gw.d/gw.d*, sendo d o número de dígitos usados para a parte fracionária do valor.

Todos os descritores podem ser precedidos por um valor inteiro que indica o número de vezes que o descritor deve ser repetido em uma linha. Por exemplo, *9f12.5* significa repetir o descritor *f12.5* nove vezes. Eles também podem ser combinados, possibilitando que diferentes combinações de valores possam ser enviados à tela ou a um arquivo.

As linhas abaixo ilustram o uso desta subrotina, inclusive comparando-o com a subrotina **print**:

```
x=(/ (/1,2/), (/3,4/) /)
write_matrix(x, "2i2", False)
```

que resulta na seguinte saída na tela:

```
1    2
3    4
```

diferentemente, a subrotina **print** imprime um elemento após o outro:

```
print(x)

Variable: x
Type: integer
Total Size: 16 bytes
            4 values
Number of Dimensions: 2
Dimensions and sizes: [2] x [2]
Coordinates:
(0,0) 1
(0,1) 2
(1,0) 3
(1,1) 4
```

Se a intenção fosse guardar os dados de $x$ em um arquivo ASCII, em sua forma tabular, devemos passar uma variável do tipo *logical* ao terceiro argumento da subrotina **write_matrix** contendo o atributo *fout*, como ilustrado nas linhas abaixo:

```
x=(/ (/1,2/), (/3,4/) /)
saida = True
saida@fout = "arquivo_dados.txt"
write_matrix(x, "2i2", saida)
```

No diretório atual será criado um arquivo chamado "arquivo_dados.txt" contendo os valores presentes em $x$.

# Capítulo 5

# Análise de Dados

O NCL oferece diferentes abordagens para análise de dados: (1) sintaxe de arranjos ,(2) centenas de funções internas, (3) funções disponibilizadas por usuários e (4) rotinas em linguagem Fortran ou C. Dicas sobre codificação eficientes podem ser encontradas em:

`htt://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclUsage.shtml`

## 5.1 Sintaxe de arranjos

A álgebra do NCL, como no Fortran 90, suporta operações sobre escalares e matrizes, ao invés de apenas entre escalares, como nas linguagens C, C++ e PASCAL. Para trabalhar com operações matricias, ambos os operandos devem ter o mesmo número de dimensões que, por sua vez, devem ter o mesmo tamanho. Além disso, os tipos de dados de cada operando devem ser equivalentes ou um operando deve ser coercível ao tipo do outro operando. Por exemplo, supondo que $A$ e $B$ sejam arranjos com mesmo número e tamanho de dimensões:

```
C=A+B     ;adição elemento por elemento
D=A-B     ;subtração elemento por elemento
E=A*B     ;multiplicação elemento por elemento
```

nas quais $C$, $D$ e $E$ serão automaticamente criados se não existiam anteriormente. Estes novos arranjos terão o mesmo número e tamanho de dimensões

dos arranjos $A$ e $B$.

**Exemplo 1:**

```
A=(/10,20,30,40,50,60/)
B=(/60,50,40,30,20,10/)
C=A+B
D=A-B
E=A*B
print(C)
print(D)
print(E)
```

Resulta em:

$C$=(70,70,70,70,70,70)
$D$=(-50,-30,-10,10,30,50)
$E$=(600,100,1200,100,600)

Os valores escalares são casos especiais, quando consideramos operações com matrizes. Quando um valor escalar aparece em uma expressão com um valor multi-dimensional (i.e., uma matriz), o valor escalar é aplicado a cada elemento da matriz. Por exemplo, considerando que $E$ é uma matriz de $n$ linhas e $m$ colunas, a seguinte operação

*F= 2*E+5*

fará com que cada elemento da matriz $E$ seja multiplicado por *2* e em seguida *5* será adicionado a cada elemento. O resultado será atribuído a $F$, que terá as mesmas $n$ linhas e $m$ colunas de $E$.

**Exemplo 2:**

```
E=(/11,22,33,44,55,66/)
F=2*E+5
print(F)
```

Resulta em:

$F$=(27,49,71,93,115,137)

Os operadores $<$ e $>$ do NCL não são comumente usados em outras linguagens (ás vezes chamados de "operadores de recortes"). O uso destes operadores é ilustrado no exemplo abaixo:

```
x=(/ (/1,2/), (/3,4/) /)
y = x > 3
print(y)
Variable: y
Type: integer
Total Size: 16 bytes
 4 values
Number of Dimensions: 2
Dimensions and sizes: [2] x [2]
Coordinates:
(0,0) 3
(0,1) 3
(1,0) 3
(1,1) 4
```

Significa que qualquer valor de $x$ superior a 3 será substituído por este último. Todas as expressões envolvendo arranjos automaticamente ignoram qualquer operação que envolva valores definidos para **_FillValue**.

## 5.2 Conformação da matriz

Expressões com arranjos exigem que todos os operandos tenham o mesmo número de dimensões e dimensões com mesmo tamanho. Valores escalares possuem conformação a qualquer arranjo. O exemplo abaixo ilustra isto

```
T=(/10,30,64,128/)
P=(/54,1,21,2/)
theta = T + (5 * P)
print(theta)
```

sendo $T$ e $P$ são vetores de mesmo tamanho e, logo, *theta* possuirá os seguintes valores (104,151,341,642), tendo o mesmo número de elementos de $T$ e $P$.

As funções **conform** e **conform_dims** podem ser usadas adaptar a forma de um arranjo a outro com diferentes dimensões e tamanhos de dimensões:

```
T=new((/5,3/),float)
P=new((/3/),float)
```

```
P=(/1000,850,700/)
T=5.
Pconform=conform(T,P,1)
```

A função **conform** expande $P$ a um arranjo bidimensional igual a $T$. Note que é preciso informar qual dimensão de $T$ tem o mesmo tamanho de $P$, que neste caso corresponde a dimensão “1” (segunda dimensão) de $T$. Para mais detalhes veja: `http://ncl.ucar.edu/Document/Functions/Built-in/conform.shtml` e `http://ncl.ucar.edu/Document/Functions/Built-in/conform_dims.shtml`.

## 5.3 Alocação de memória com arranjos

Memória pode ser explicitamente reservada ou criada para arranjos de duas formas diferentes:

1. Com o construtor de matriz (/.../):

```
a_integer   =   (/1.0, 2.0, 3.0/)
a_float     =   (/1.0, 2.0, 3.0/)
a_double    =   (/4d0,5d-5,1d30/)
a_string    =   (/“a”,“b”,“c”/)
a_logical   =   (/True, False ,True/)
a_2darray   =   (/ (/1,2/) , (/5,6/) , (/8,9/) /)
```

2. Com a função **new**(tamaho_arranjo,formato,tipo,[_*FillValue*]): A inclusão de _*FillValue* é opcional e caso ele não estiver presente, será atribuído o valor padrão (ver Apêndice D). Especificando “**No_FillValue**” nenhum _*FillValue* padrão será atribuído. Veja os exemplos abaixo:

```
a=new(3,float)                            ;   999.
b=new(10,float,1e20)                      ;   1e20
c=new((/5,6,7/),integer)                  ;   999
d=new( dimsizes(U)), double)              ;   9999
e=new( dimsizes(ndtooned(U)),logical)     ;   -1
s=new(100,string)                         ;   “missing"
q=new(3,float,“No_FillValue”)             ;   nenhum_FillValue”
```

A memória é automaticamente criada por funções de variáveis retornáveis, assim, a utilização de **new** muitas vezes não é necessária ou recomendada.

## 5.4 Funções e subrotinas

Como a maioria das linguagens, o NCL possui funções e subrotinas. Há diferenças entre elas que os usuários devem estar cientes:

1. Funções são porções de códigos na linguagem NCL que retornam um ou mais valores e podem ser usadas como parte de uma expressão. Por exemplo, **max**, **sin** e **exp** são funções matemáticas que retornam o valor máximo, o seno e o exponencial do argumento passado a elas. O exemplo seguinte ilusta o uso de funções em expressões matemáticas

   z= **exp** ( **sin**(**max** (q) ) ) + 12.345

   Funções não são obrigadas a devolver o mesmo tipo e tamanho de matriz cada vez que eles são chamados.

2. Subrotinas (análogas a subrotinas do Fortran) não podem fazer parte de uma expressão, porque eles não retornam valores. As subrotinas do NCL são usadas de forma semelhante às utilizadas em outras linguagens de programação. Elas executam uma tarefa específica e/ou são usadas para modificar um ou mais dos argumentos de entrada.

#### Argumentos, procedimentos e funções para o NCL

Os argumentos são passados às funções e subrotinas por referência. Isso significa que mudanças em seus valores, atributos, nomes de dimensão e coordenadas das variáveis dentro de uma função ou procedimento irá provocar uma mudança em seus valores no programa principal. Por convenção, então, os argumentos para as funções não devem ser alterados por uma função, embora isso não seja necessário. Na discussão a seguir, será assumido que os argumentos para funções seguem esta convenção.

#### Protótipo de um argumento

Em NCL os argumentos de uma função e subrotinas podem ser especificados para serem muito restritos e requerem um tipo específico, número de dimensões e um tamanho para cada dimensão. Eles podem não ter restrições de tipo ou dimensão. A estas especificações de argumento chama-se de protótipo de argumento.

1. Especificação restrita de argumentos significa que os argumentos são obrigados a ter um determinado tipo, tamanho e forma de dimensão:

**procedure** ex (x[*][*]:**float**, y[2]: **byte**,*res*: **logical**, *text* :**string**)

O argumento $x$ deve ser um arranjo bidimensional do tipo **float**, $y$ deve ser um arranjo unidimensional de comprimento 2 ,*res* e *text* devem ser do tipo **logical** e **string** respectivamente, mas podem ser de qualquer dimensionalidade.

2. Tipo genérico de prototipagem: único tipo

   **function** *xy_interp* (*x*: **numeric** ,*y*: **numeric**)

   Aqui **numeric** é qualquer tipo de dado numérico indicado.

3. Sem tipo, sem tamanho, sem especificação de forma tridimensional:

   **procedure** *foo (a,b,c)*

4. Combinação:

   **function** *ex (d[*]* :**float**,*x* : **numeric**, *wks*: **graphic** , y*[2],a*)

Uma característica muito importante que os usuários devem estar cientes é quanto a passagem de argumentos para subrotinas. Isto é um problema apenas quando espera-se que a subrotina modifique os argumentos de entrada. Quando um argumento de entrada deve ser forçado para o tipo correto da subrotina, o NCL não é capaz de transformar os dados na direção inversa, de modo que o argumento não é afetado pelas alterações feitas no âmbito da subrotina. O NCL gera uma mensagem de aviso, uma mensagem *WARNING*.

# Capítulo 6

# Linhas de comando

O NCL suporta um numero limitado de opções e definições, além da definição e execução de comandos simples do NCL ainda no terminal do Linux, independendo do modo de uso do NCL: interativo ou em lote. Maiores detalhes e exemplos são apresentados descritos no site oficial do NCL: `http://www.ncl.ucar.edu/Document/Manuals/RefManual/NclCLO.shtml`.

## 6.1 Alterando o comportamento do NCL

Ao executar o NCL no terminal do Linux é possível passar opções que alteram o seu comportamento. São elas:

| | |
|---|---|
| -h | Uma ajuda rápida, mostra as opções disponíveis. |
| -n | Não enumera os valores na impressos com a subrotina **print**. |
| -x | Repete os comandos do NCL usados no modo interativo. |
| -V | Apenas mostra a versão do NCL. |

Um exemplo simples usando a função *-x*:

```
$ ncl -x
 Copyright (C) 1995-2011 - All Rights Reserved
 University Corporation for Atmospheric Research
 NCAR Command Language Version 6.0.0
 The use of this software is governed by a License Agreement.
 See http://www.ncl.ucar.edu/ for more details.
```

```
ncl 0> a = 5
+ a = 5
ncl 1> exit
 + exit
```

Note que a primeira linha refere-se ao terminal do Linux, no qual executa-se o NCL com a opção *-x*. Além disso, devido ao uso desta opção, cada linha digitada no NCL é repetida logo abaixo.

## 6.2 Especificando atribuições de variável na linha de comando

Criar variáveis na linha de comando quando o NCL é executado pode facilitar as tarefas de processamento de dados. Espaços não são permitidos. Declarações contendo sequências devem ser colocadas entre aspas simples. O script pode conter configuração padrão para variáveis que são opcionais. Veja alguns exemplos:

```
$ ncl c=5
 Copyright (C) 1995-2011 - All Rights Reserved
 University Corporation for Atmospheric Research
 NCAR Command Language Version 6.0.0
 The use of this software is governed by a License Agreement.
 See http://www.ncl.ucar.edu/ for more details.
ncl 0> if(c.gt.0) then
ncl 1> print("c eh maior que zero")
ncl 2> else
ncl 3> print("c eh menor ou igual a zero")
ncl 4> end if
(0) c eh maior que zero
```

Veja que foi criada uma variável *c* cujo valor é igual a 5. Ela foi usada num teste com um **if ... end if**. No exemplo seguinte, um vetor de dados é criado na linha de comando do Linux e usado dentro do NCL:

```
$ ncl 'dados=(/1900,1920,1940,1960,1980,2000/)'
 Copyright (C) 1995-2011 - All Rights Reserved
 University Corporation for Atmospheric Research
 NCAR Command Language Version 6.0.0
```

```
 The use of this software is governed by a License Agreement.
 See http://www.ncl.ucar.edu/ for more details.
ncl 0> print(dados)

Variable: dados
Type: integer
Total Size: 24 bytes
            6 values
Number of Dimensions: 1
Dimensions and sizes: [6]
Coordinates:
(0) 1900
(1) 1920
(2) 1940
(3) 1960
(4) 1980
(5) 2000
```

Da mesma forma, é possível criar uma variável na linha de comando do NCL, usando-a com um script em NCL. Veja o exemplo seguinte:

```
$ ncl anoInicio=1800 anoFim=2005 meuScript.ncl
```

Neste exemplo, foram criadas duas variáveis, *anoInicio* e *anoFim*, que serão usadas pelo script em NCL *meuScript.ncl*.

# Capítulo 7

# Usando códigos externos

O NCL. que foi escrito em C, foi desenvolvido para permitir que os seus usuários chamem códigos externos (Fortran ou outras bibliotecas, por exemplo). O principal foco aqui é o uso de subrotinas em Fortran (77 ou 90). Para usar funções externas em linguagem C veja: `http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclExtend.shtml`.

## 7.1 Interface NCL/Fortran

O uso de subrotinas em Fortran é facilitada pelo utilitário *WRAPIT*, que é distribuído com o NCL. As opções disponíveis pode ser vistas com *WRAPIT -h*. O utilitário *WRAPIT* compila o código externo e gera um arquivo que, por convenção, é chamado de "objeto compartilhado" (*shared object*, em inglês). Este objeto é identificado por uma extensão ".so". A única informação que *WRAPIT* necessita é a interface entre o Fortran e o NCL, incluindo a declaração da subrotina e de seus argumentos. A especificação explícita dos tipos dos argumentos não é necessária, pois o *WRAPIT* está ciente dos tipos padrões do Fortran. O NCL usa um par delimitador como interface:

**C NCLFORTSTART**
**C NCLEND**

para identificar a seção de interface. Note que os delimitadores estão na forma de comentários em Fortran 77 e, assim, não têm efeito sobre o código. O **C NCLFORTSTART** precede a declaração da subrotina enquanto que **C**

**NCLEND** segue a última declaração de argumentos pertencentes à interface. Exemplo:

```
C NCLFORTSTART
   subroutine demo(xin,xout,mlon,nlat,text)
   integer mlon, nlat
   real xin(mlon,nlat), xout(mlon,nlat)
   character*(*) text
C NCLEND
```

## 7.2 Subrotinas em Fortran 77

Quatro passos são necessários para criar e chamar objetos compartilhados. Eles serão ilustrados por um exemplo: considere um arquivo fonte chamado *foo.f*. Este arquivo pode conter uma ou mais subrotinas escritas em Fortran 77.

**Passo 1**: coloque cada subrotina entre os delimitadores de interface:

```
C NCLFORTSTART
   subroutine demo(xin,xout,mlon,nlat,text)
   integer mlon, nlat
   real xin(mlon,nlat), xout(mlon,nlat)
   character*(*) text
C NCLEND
```

O resto do código em Fortran pode conter muitas subrotinas.

**Passo 2**: crie um objeto compartilhado usando **WRAPIT**. A linha de comando

```
WRAPIT foo.f
```

criará um arquivo chamado foo.so, ou seja, o nome do arquivo compartilhado será o mesmo do arquivo Fortran 77, acrescido da extensão ".so".

**Passo 3**: adicione uma declaração **external** ao script NCL. A declaração **external** consiste de um identificador arbitrário, que o NCL usa para selecionar dinamicamente o objeto compartilhado externo (o mais comum é usar o nome do arquivo Fortran) e o local do objeto compartilhado. O local padrão é o diretório atual,

**external** FOO "./foo.so"

**Passo 4**: chame a rotina desejada a partir do script NCL. Há uma sintaxe específica que deve ser usada, que inclui (a) o nome pelo qual o NCL identifica o objeto compartilhado, (b) o separador :: e (c) a interface da subrotina Fortran,

FOO::demo(xin,xout,nlon,nlat,text)

## 7.3 Subrotinas em Fortran 90

Chamar subrotinas em Fortran 90 requer, essencialmente, o mesmo processo usado para as subrotinas em Fortran 77, exceto pelo primeiro passo. Com subrotinas em Fortran 77, os delimitadores de interface do NCL são inseridos diretamente nas subrotinas em Fortran 77. Infelizmente, o analisador sintático ("parser", em inglês) Fortran usado pelo *WRAPIT* não entende a sintaxe do Fortran 90. Então, o usuário deve criar um *esboço* de interface ("stub interface", em inglês[1]) para cada subrotina chamada pelo NCL. Estes arquivos "stub" são uma repetição da lista de declarações Fortran 90, na sintaxe do Fortran 77. Não há necessidade de que estes arquivos sejam subrotinas completas. Lembre-se: o *WRAPIT* apenas trata a chamada da subrotina e os seus argumentos. Considere a seguinte subrotina contida num arquivo chamado "soma.f90":

```
subroutine soma2numeros(x,y,z)
   implicit none
   real, intent(in) :: x,y
   real, intent(out) :: z
   z = x + y
   return
end subroutine soma2numeros
```

Para usá-la com o NCL, deve-se:

**Passo 1**: criar um esboço de interface usando a sintaxe do Fortran 77 e armazená-lo em um arquivo chamado *soma90.stub*. Cada arquivo requer um conjunto de delimitadores **C NCLFORTSTART** e **C NCLEND**:

[1]Um "stub" é um termo da área de desenvolvimento de software que refere-se a um pedaço de código usado para substituir algumas outras funcionalidades de programação. Mais em `http://pt.wikipedia.org/wiki/Stub`.

```
C NCLFORTSTART
   subroutine soma2numeros(x,y,z)
   real x,y,z
C NCLEND
```

**Passo 2**: criar o objeto compartilhado usando o *WRAPIT*. Se módulos são usados no código em Fortran 90, eles devem ser compilados antes das rotinas que usam-nos. O usuário deve especificar o compilador a ser usado na linha de comando. Este passo é realizado com a linha de comando

```
WRAPIT soma90.stub soma.f90
```

**Passos 3 e 4**: iguais aos realizados para subrotinas em Fortran 77.

Um script NCL que usaria estas subrotinas em Fortran 90 seria

```
external SOMA2NRO "./soma2nros.so"
begin
   a=10.
   b=20.
   c=new(1,float)
   SOMA2NRO::soma2numeros(a,b,c)
   print("A soma eh "+c)
end
```

## 7.4 Usando bibliotecas comerciais

O processo é similar aquele usado com códigos em Fortran 90, pois o usuário deve criar um arquivo “stub” para especificar explicitamente a sequência de chamada necessária e os tipos de argumentos. Assuma que se quer usar a subrotina *rcurv* da biblioteca IMSL. Por conveniência, a sintaxe do Fortran 77 será usada:

**Passo 1**: crie um programa “invólucro” arbitrariamente chamado de *rcurvWrap.f*:

```
C NCLFORTSTART
   subroutine rcurvwrap(n,x,y,nd,b,s,st,n1)
   integer n, nd, n1
   real x(n),y(n),st(10),b(n1),s(n1)
```

```
C NCLEND
   call rcurv(n,x,y,nd,b,s,st) ! nome na IMSL
   return
   end
```

**Passo 2**: use o *WRAPIT* para compilar a subrotina "invólucro" e para especificar o local da biblioteca IMSL para o sistema local:

```
WRAPIT -l mp -L/usr/local/lib64/r4i4 -l imsl_mp rcurvWrap.f
```

**Passos 3 e 4**: iguais aos das duas seções anteriores.

Agora, um script exemplo, que usaria este subrotina:

```
external IMSL "./rcurvWrap.so"
begin
   x = (/0,0,1,1,2,2,4,4,5,5,6,6,7,7/)
   b = (/508.1,498.4,568.2,577.3,651.7,657.0,\
    755.3,758.9,787.6,792.1,841.4,831.8,\
    854.7,871.4/)
   nobs = dimsizes(y)
   nd = 2
   n1 = nd+1
   b = new(n1,typeof(y))
   s = new(n1,typeof(y))
   st = new(10,typeof(y))
   IMSL::rcurvwrap(nobs,x,y,nd,b,s,st,n1)
   print(b)
   print(s)
   print(st)
end
```

## 7.5 O que o *WRAPIT* faz

O *WRAPIT* é um script UNIX/Linux que desempenha várias tarefas incluindo a compilação em Fortran e geração de um objeto compartilhado (.so). Ele disponibiliza várias opções aos usuários; veja-as digitando num terminal do UNIX/Linux

```
WRAPIT -h
```

O *WRAPIT* faz as seguintes tarefas:

1. Usa um utilitário do NCL chamado *wrapit77*, um programa em linguagem C, para criar um programa "invólucro" em linguagem C que chama o analisador gramatical (ou "parser", em inglês) de Fortran 77 e cria o código em linguagem C necessário à interface entre NCL e Fortran.

   *wrapit77 < foo.f >! foo_W.c*

2. Compila e cria os objetos para os códigos em linguagens C e Fortran:

   *cc -c foo_W.c ; foo_W.o*
   *f77 -c foo.c ; foo.o*

3. Cria um objeto compartilhado dinâmico (.so) usando o utilitário local *ld*.

4. Apaga os arquivos temporários para que reste apenas o arquivo do objeto compartilhado (.so).

## 7.6 Mapeamento de arranjos NCL/Fortran

Em Fortran, a dimensão mais à esquerda de um arranjo (as linhas de uma matriz, por exemplo) é aquela que varia mais rapidamente, enquanto que no NCL é a dimensão mais à direita que varia mais rapidamente. Em algumas situações, isto pode causar confusão. Apesar disto, raramente se faz necessário a reordenação de um arranjo ao chamar um subrotina escrita em Fortran, a partir de um script em NCL. Então, mesmo sabendo que os nomes das dimensões aparecem em ordem inversa, os elementos dos arranjos mapeiam-se diretamente. A regra "as dimensões de variação mais rápida em uma linguagem mapeiam à dimensão de variação mais rápida em outra linguagem" aplica-se aqui:

| NCL | | Fortran |
|---|---|---|
| x(time,lev,lat,lon) | <=map=> | x(lon,lat,lev,time) |

Considere agora dois arranjos com N=2 e M=3:

ncl: x(N,M) <==> x(M,N) : Fortran

x(0,0) <==> x(1,1)
x(0,1) <==> x(2,1)

x(0,2) <==> x(3,1)
x(1,0) <==> x(1,2)
x(1,1) <==> x(2,2)
x(1,2) <==> x(3,2)

## 7.7 NCL e Fortran (ou C) em um script Shell

Ao trabalhar com um script em NCL que chama um ou mais objetos compartilhados (.so) em Fortran (ou em C), é conveniente combinar os vários passos em um único script em Shell, do UNIX/Linux. Abaixo está um esboço de um script em C-Shell contendo um exemplo do uso do NCL e do Fortran, utilizando os códigos apresentados na seção 7.3:

```
#!/usr/bin/csh
# ===== Código em NCL ========
cat >! main.ncl « "END_NCL"
external SOMA2NRO "./soma2nros.so"
begin
   a=10.
   b=20.
   c=new(1,float)
   SOMA2NRO::soma2numeros(a,b,c)
   print("A soma eh "+c)
end
"END_NCL"

# ==== Código em Fortran 90 ======
cat >! soma2nros.f90 « "END_F90"
subroutine soma2numeros(x,y,z)
   implicit none
   real, intent(in) :: x,y
   real, intent(out) :: z
   z = x + y
   return
end subroutine soma2numeros
"END_F90"

# ==== Arquivo STUB para Fortran 90 ==
cat >! soma2nros.stub « "END_STUB"
```

```
C NCLFORTSTART
   subroutine soma2numeros(x,y,z)
   real x,y,z
C NCLEND
"END_STUB"

# ===== Chama o WRAPIT =====
WRAPIT soma2nros.stub soma2nros.f90

# ===== Executa o código em NCL =====
ncl main.ncl >&! main.out
exit
```

# Capítulo 8

# Gráficos

O NCL possui três áreas distintas. A primeira relaciona-se à leitura e à escrita de arquivos, visto aqui, no Capítulo 3. O NCL disponibiliza uma sintaxe única, que suporta o referenciamento direto à variáveis contidas em arquivos. Isto possibilita que não apenas uma variável ser lida ou escrita inteiramente, mas que porções dela possam ser acessadas, por meio das regras de indexação de variáveis de arquivo. O acesso à informações adicionais sobre as variáveis, chamadas de *metadados*. Normalmente, estes metadados fornecem informações a respeito das coordenadas de grade, unidades, valores ausentes (_*FillValue*) entre outras.

A segunda área refere-se ao processamento de dados. O aprendizado desta área depende da complexidade das necessidades de processamento de dados do usuário.

Finalmente, a geração de gráficos refere-se à última área. As sintaxes dos comandos gráficos são, normalmente, simples, mas quem acaba definindo a verdadeira complexidade é a aplicação dada pelo usuário. O NCL usa a Biblioteca de Utilidades de Alto Nível do NCAR Graphics (HLUs) para configurar a saída gráfica. Uma breve discussão sobre como criar gráficos é dada neste capítulo.

É impossível abordar todos os tipos de gráficos que podem ser produzidos pelo NCL. Aqui, é dada maior ênfase ao conceito de geração de gráficos, buscando evidenciar os pontos mais importantes e necessários à criação de qualquer tipo de gráfico. O leitor é convidado a visitar a seção de exemplo do site oficial do NCL - `http://www.ncl.ucar.edu/Applications/` - para conhecer melhor as potencialidades gráficas do NCL.

## 8.1 Um script simples

A criação de gráficos pode tornar-se muito complexa, dependendo de como se quer plotar os dados em mãos. Em termos gerais, há cinco passos básicos necessários para se criar um gráfico simples:

1. Abra uma área de trabalho gráfica, ou *workstation*
2. Descreva os seus dados
3. Crie o gráfico desejado
4. Desenhe o gráfico
5. Chame a subrotina *frame*

O script em NCL abaixo mostra como plotar um gráfico XY simples, apresentado na Figura 8.1, que servirá para ilustrar os passos descritos acima.

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin

   wks = gsn_open_wks( "x11", "xyPlot")

   y = random_uniform(-10,10,100)

   plotXY = gsn_csm_y( wks, y, False )

end
```

As duas primeiras linhas deste script carregam à memória as interfaces e rotinas gráficas de alto nível que são distribuídas com o NCL. Embora as funções e subrotinas incluídas nas bibliotecas possam ser carregadas a qualquer instante antes de seu uso, usa-se, como convenção, o carregamento delas no inicio do script, antes da linha *begin*.

Os gráficos do NCL são baseados em métodos orientados a objetos (OO). Esta abordagem fornece poder e flexibilidade consideráveis, mas pode ser muito fatigante e tediosa. Para ajudar os usuários, dois conjuntos de interfaces gráficas de alto nível foram desenvolvidas. Estas interfaces facilitam o
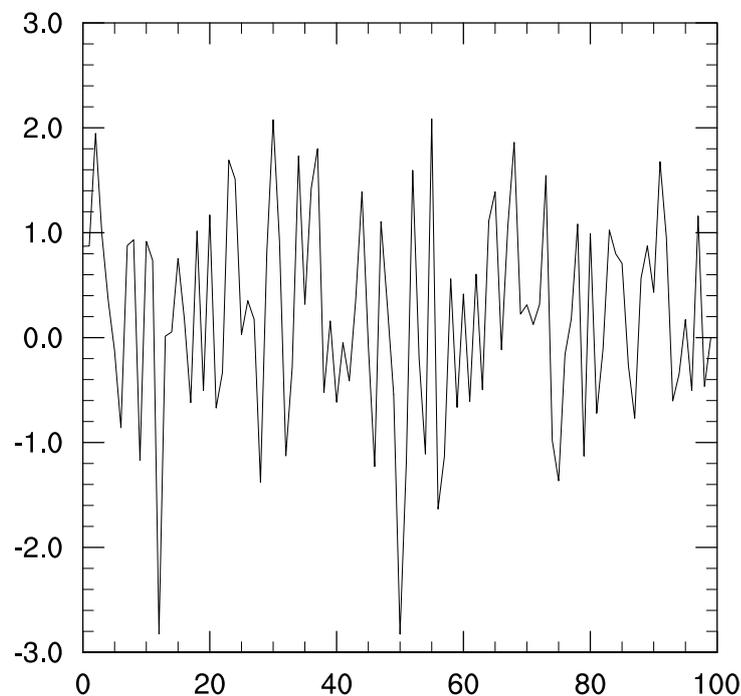
Figura 8.1: *Gráfico resultante do script NCL*

processo de visualização, retendo os benefícios da abordagem OO. Por razões históricas, todas as interfaces de plotagem começam com *gsn_*, que signifca "*Getting Started with NCL*".

A primeira linha após *begin* realiza o primeiro passo: cria uma área de trabalho gráfica. A função *gsn_open_wks* pede dois argumentos (i) o tipo da área gráfica e (ii) o título dela. O tipo da área gráfica pode ser dos seguintes (a) um arquivo NCGM ("ncgm"), (b) um arquivo PostScript ("ps", "eps"ou "epsi"), (c) um arquivo PDF ("pdf") ou (d) uma janela X11 ("x11"). Neste exemplo, uma janela X11 com título *xyPlot* será criada. Note que o nome desta variável é uma abreviatura de *workstation*, termo dado à área de trabalho gráfica, mas não é obrigatório o uso deste nome, apesar dele estar presente nos inúmeros exemplos disponíveis no site oficial do NCL.

A linha seguinte atribui a variável $y$ um vetor com 100 valores aleatórios dentro do intervalo $[-10; 10]$, criado por um gerador de números aleatórios baseado em uma distribuição uniforme, com a ajuda da função **ran-**

**dom_uniform**. Pode-se dizer que temos aqui o passo 2, ou seja, a descrição dos nossos dados, que foram atribuídos a uma variável do NCL.

Na linha posterior cria-se o gráfico deseja, ou seja, o nosso 3 passo. Nesta linha, é criada, por atribuição, uma variável do tipo *graphic* que armazenará o nosso gráfico. O gráfico é criado por meio da função gráfica **gsn_csm_y**, um gráfico do tipo XY. Esta função, para funcionar, precisa de três informações: (1) onde desenhar o gráfico; (2) que dados usar e (3) onde estão as opções para modificação do gráfico, caso sejam definidas.

Para indicar onde desenhar, basta passar à função a variável do NCL com a qual criou-se a área de trabalho gráfica, neste caso, a variável *wks*, criada no primeiro passo. Os dados que serão usados para o gráfico são passados à função por meio da variável $y$. Neste exemplo, nenhuma opção de modificação do gráfico é passada. Portanto, o gráfico será criado em sua configuração padrão. A estas opções é dado o nome de recursos gráficos (*resources*, em seu termo original, em inglês). Elas serão melhor discutidas na seção seguinte.

Os últimos dois passos para a criação do nosso gráfico são realizados automaticamente pela função **gsn_csm_y**. Em certas situações é necessário alterar este comportamento padrão. Isto é feito por meio de recursos gráficos específicos.

Pronto! Nossos dados foram plotados num gráfico XY, no qual os valores dos elementos da variável $y$ está associada a sua posição no vetor, ou seja, o primeiro valor do vetor $y$ é posicionado em $x = 0$, o segundo em $x = 1$ e assim por diante, até o último valor, que será posicionado em $x = 99$, conforme mostra a Figura 8.1.

## 8.2 Recursos gráficos (*resources*)

As funções gráficas têm um comportamento padrão para a geração de gráficos. Dependendo do tipo de gráfico e das necessidades de personalização pode ser necessário alterar este comportamento padrão. Os recursos gráficos proporcionam o meio pelo qual isto é feito. Mas, o que são os recursos gráficos? Como defini-los? Como passá-los às funções gráficas?

Os recursos gráficos são simplesmente atributos. Isto posto, temos logo em mente que eles são definidos com a ajuda do símbolo “@”. A pergunta que surge em sequência é: se são atributos, são atributos de que variável? Os

recursos gráficos são atributos de uma variável do tipo *logical*, que possui um valor *True*. Para ilustrar isto, vamos modificar o nosso script anterior para incluir títulos nos eixos X e Y e para o gráfico:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin

   wks = gsn_open_wks( "x11", "xyPlot")

   y = random_uniform(-10,10,100)

   opcoes = True
   opcoes@tiMainString = "Grafico exemplo!"
   opcoes@tiXAxisString = "eixo X - indice do vetor"
   opcoes@tiYAxisString = "eixo Y - valores aleatorios"
   opcoes@gsnMaximize = True

   plotXY = gsn_csm_y( wks, y, opcoes )

end
```

Neste script criamos uma nova variável, chamada *opcoes*, cujo valor é *True*, sendo uma variável do tipo *logical*. Nas quatro linhas seguintes, vários atributos são passados a esta nova variável. Os três primeiros atributos referem-se aos títulos do gráfico, do eixo X e do eixo Y, respectivamente. Já o último permite aumentar o tamanho gráfico, que por padrão é criado num tamanho menor. Uma característica interessante e bastante útil dos recursos gráficos pode ser notada em seus nomes. Os três atributos que definem títulos aos gráficos tem seus nomes iniciados por *ti*, abreviatura de *title*, ou seja, título, em inglês. Assim, é intuitivo esperar que todos os demais atributos relacionados a títulos comecem com os mesmos dois caracteres dos usados neste exemplo. Dê uma olhada nos recursos gráficos existentes no site oficial do NCL: `http://www.ncl.ucar.edu/Document/Graphics/Resources/`.

Para usar os recursos definidos à variável *opcoes* é necessário passá-la à função **gsn_csm_y**, conforme é mostrado na penúltima linha do script. Como a função entende que a variável *opcoes* possui recursos? Simples, a função (e qualquer função gráfica) verifica se a variável lógica possui um valor igual a *True*, ou seja, verdadeiro. Em caso positivo, ela entende que há uma variável com recursos e verifica quais foram definidos. Aqueles aplicáveis ao tipo de gráfico que está sendo desenhado são usados, enquanto que os demais são

ignorados. Caso negativo, ou seja, se a variável *opcoes* tivesse um valor *False* a função gráfica ignoraria todo e qualquer recurso passado a ela. A Figura 8.2 exibe o gráfico gerado pelo script acima.
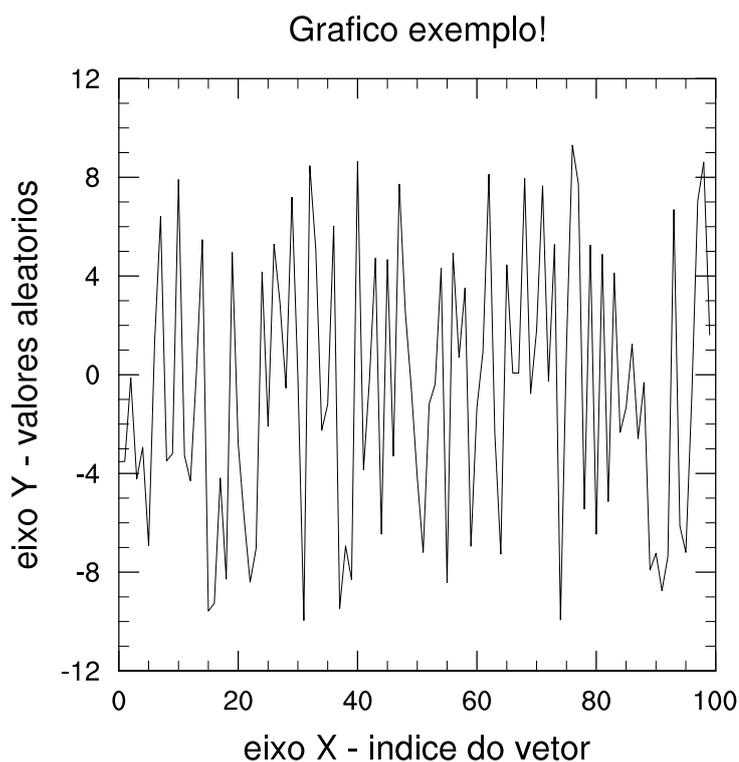
Figura 8.2: *Gráfico ilustrando o uso de recursos.*

## 8.3 Plotando campos meteorológicos

Gráficos XY são os mais simples gerados pelo NCL. Na Meteorologia, a análise de variáveis meteorológicas em uma certa região espacial é uma prática comum e importante para o entendimento dos fenômenos atmosféricos e à previsão de tempo. A distribuição espacial de variáveis meteorológicas são normalmente encontradas em arquivos de dados, sejam eles em formatos binários ou em formato ASCII. O acesso a ambos tipos de arquivos já foi tratado no Capítulo 3 e não serão discutidos neste capítulo. Para facilitar o entendimento da plotagem de campos, vamos usar o script abaixo:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin

   ; variaveis e parametros
   nx = 10
   ny = 10
   pi = acos(-1.)
   pnmm = new( (/ny,nx/), float )

   ; variacao dos eixos x e y
   dx = 2*pi/nx
   dy = 2*pi/ny

   ; gerando campo ficticio
   do j=0,ny-1
      do i=0,nx-1
         pnmm(j,i) = sin(i*dx)*cos(j*dy)*20.+1010.
      end do
   end do

   ; ambiente gráfico
   wks = gsn_open_wks( "x11", "campoFicticio")

   ; recurso gráfico
   opcoes = True
   opcoes@gsnMaximize = True
   opcoes@tiMainString = "Campo ficticio - PNMM [hPa] - NCL"

   ; plotando gráfico
   plot = gsn_csm_contour( wks, pnmm, opcoes )

end
```

Vamos discutir sobre este script se referenciando a blocos de linhas de comandos. O primeiro bloco após o comando **begin** define três variáveis escalares, que possuem parâmetros que serão usados no script, e um arranjo bidimensional, que armazenará o campo fictício de pressão ao nível médio do mar (PNMM) que será plotado. O segundo bloco define outras duas variáveis

escalares, que serão usadas para a criação do campo fictício.
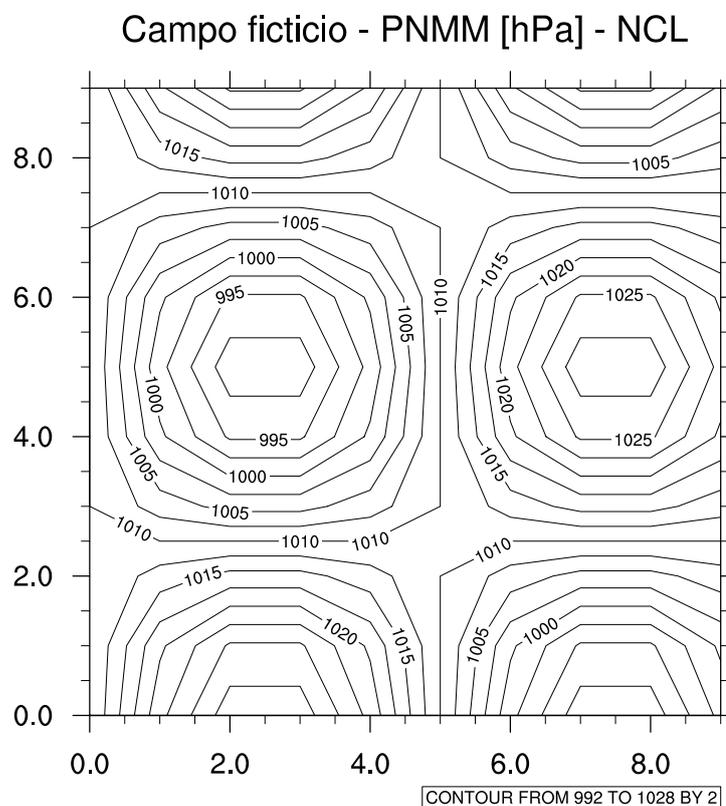
Figura 8.3: *Campo fictício de PNMM. Veja o texto para maiores detalhes.*

O terceiro bloco de linhas de comando cria o campo fictício de PNMM usando dois laços de repetição **do ... end do** vistos na seção 2.6.1 do Capítulo 2. Cada "loop" **do ... end do** é associado a uma dimensão da variável *pnmm*, que estão associadas às dimensões espaciais do campo; latitude e longitude, por exemplo. Note que temos um "loop" dentro de outro. A esta configuração dá-se o nome de "loops" aninhados, sendo aquele mais "interno" o de mais rápida variação. Em suma, os "loops" permitem o acesso a pontos distintos no espaço, com coordenadas $(j, i)$, aos quais são atribuídos valores fictícios de PNMM dados pela função usada neste bloco.

Nos blocos seguintes, não temos nada de novo, exceto pelo uso de uma nova função: **gsn_csm_contour**, apropriada para o desenho de campos. Neste caso, o nome da função é bastante intuitivo, pois a palavra "contour" presente em seu nome refere-se a contorno, indicativo de seu objetivo: desenhar a distribuição de uma variável por meio de isolinhas ou contornos. A Figura 8.3 mostra o resultado deste script.

## 8.4 As funções draw e frame

Foi mencionado anteriormente que as duas etapas finais da criação de um gráfico no NCL são feitas automaticamente pelas funções de plotagem. Nos exemplos anteriores, as funções **gsn_csm_y** e **gsn_csm_contour**. Quando pode ser necessário mudar este comportamento? Um exemplo simples e que é abordado aqui é a inclusão de mais de uma linha num gráfico XY. Para ilustrar isso, vamos usar o mesmo exemplo inicial apresentado neste capítulo. No script abaixo, temos um vetor de dados adicional e queremos adicioná-lo ao gráfico já feito.

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin

   wks = gsn_open_wks( "x11", "xyPlot")

   y = random_uniform(-10,10,100)
   y2 = random_normal(0,1,100)

   opcoes = True
   opcoes@tiMainString = "Grafico exemplo!"
   opcoes@tiXAxisString = "eixo X - indice do vetor"
   opcoes@tiYAxisString = "eixo Y - valores aleatorios"
   opcoes@gsnMaximize = True
   opcoes@gsnFrame = False
   opcoes@gsnDraw = False

   plotXY = gsn_csm_y( wks, y, opcoes )
   linha = gsn_add_polyline( wks, plotXY, ispan(0,99,1), y2, False )

   draw(plotXY)
   frame(wks)

end
```

Aqui temos duas alterações significativas, além da inclusão de um outro vetor de dados, *y2*: (1) o uso dos atributos *gsnFrame* e *gsnDraw*, ambos com o mesmo valor lógico *False* e (2) o uso de uma nova função de plotagem, **gsn_add_polyline**, que adiciona a nova linha ao gráfico. Os atributos *gsnDraw = False* e *gsnFrame = False* avisam o gráfico gerado pelas fun-

ções **gsn_*** não devem ser desenhadas instantaneamente e que a página de plotagem não deve ser finalizada, respectivamente.
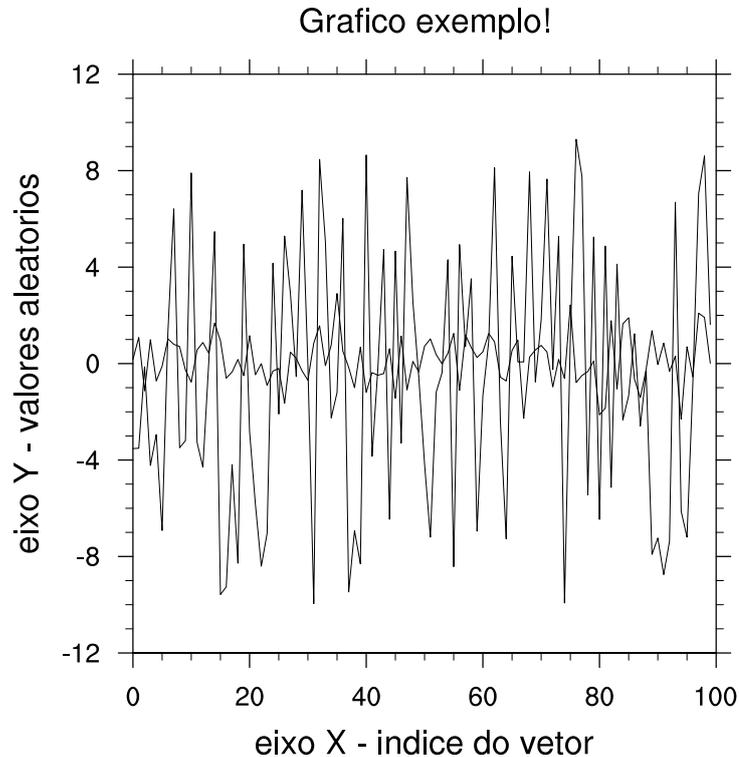
Figura 8.4: *Gráfico XY que ilustra a plotagem de duas linhas, separadamente. Veja o texto para maiores detalhes.*

Aqui deve-se esclarecer um conceito importante: uma área de trabalho gráfica (a “workstation”) pode conter várias páginas, cada uma com um gráfico diferente. A estas páginas dá-se o nome de “frame”. Assim, por definição, ao se usar qualquer uma das funções **gsn_***, o gráfico será desenhado e a página ou “frame” será finalizada. Podemos montar uma analogia disto com um bloco de desenho. Ao desenharmos numa página, viramos a página para desenhar um novo desenho. Claro que, dependendo do desenho, podemos desenhá-lo por partes. Desenhamos uma parte e não viramos a página, para terminar o desenho.

Após usar a função **gsn_csm_y**, que define qual gráfico será gerado, usamos a função **gsn_add_polyline**, que adiciona uma linha ao gráfico. Note que é necessário passar 5 argumentos a esta função. O primeiro indica em qual área de trabalho gráfica esta linha será desenhada (wks), o segundo indica em

qual gráfico a linha será adicionada (plotXY), o terceiro e o quarto referem-se às coordenadas X e Y da linha a ser desenhada, respectivamente. Por fim, o último argumento refere-se aos recursos gráficos que modificam o desenho padrão de uma linha. Neste exemplo, nenhum é passado à função.

Para finalizar, usa-se as funções **draw** e **frame** para desenhar o gráfico que encontra-se na variável *plotXY* e para finalizar a página de plotagem da área de trabalho gráfica, que está definida pela variável *wks*, respectivamente. A Figura 8.4 apresenta o resultado do script discutido nesta seçao.

Apesar deste exemplo simples poder ser feito de uma maneira mais simples (fica o desafio ao usuário!) pode-se facilmente extrapolá-lo para outras situações como um campo meteorológico, no qual pode-se querer incluir uma linha para indicar onde será feito um determinado corte vertical, por exemplo, ou um retângulo para indicar a área de maior interesse para uma determinada análise.

## 8.5 Campos meteorológicos sobre mapas

Na seção 8.3 viu-se a plotagem de campos meteorológicos, usando um campo fictício de PNMM. Entretanto, o campo gerado não foi posicionado sobre qualquer mapa. Ou seja, ao campo não foi dada nenhuma coordenada de localização geográfica. Para que um campo seja posicionado sobre alguma região geográfica deve-se ter

1. coordenadas associadas às dimensões do arranjo que se referem às posições espaciais, normalmente latitude e longitude, dos dados.
2. uma função que plote o campo sobre um mapa

O primeiro item é fácil de resolver, afinal de contas, já sabemos como atribuir nomes e coordenadas às dimensões de um arranjo, conforme explicado no Capítulo 2. O segundo item também é fácil de resolver, bastando procurar no conjunto de funções do NCL aquela que oferece esta capacidade. Neste caso, usaremos a função **gsn_csm_contour_map**. Para ilustrar o uso de mapas, vamos nos basear no mesmo script apresentado na seção 8.3:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
```

```
begin

   ; variaveis e parametros
   nx = 10
   ny = 10
   pi = acos(-1.)
   pnmm = new( (/ny,nx/), float )

   ; variacao dos eixos x e y
   dx = 2*pi/nx
   dy = 2*pi/ny

   ; gerando campo ficticio
   do j=0,ny-1
      do i=0,nx-1
         pnmm(j,i) = sin(i*dx)*cos(j*dy)*20.+1010.
      end do
   end do

   ; atribuindo coordenadas aos dados
   pnmm!0 = "lat"
   pnmm!1 = "lon"
   pnmm&lat = fspan(-35,-45,10)
   pnmm&lon = fspan(-55,-45,10)
   pnmm&lat@units = "degrees_north"
   pnmm&lon@units = "degrees_east"

   ; ambiente gráfico
   wks = gsn_open_wks( "x11", "campoFicticio")

   ; recursos gráficos
   opcoes = True
   opcoes@gsnMaximize = True
   opcoes@tiMainString = "Campo ficticio - PNMM [hPa] - NCL"
   opcoes@gsnAddCyclic = False
   opcoes@mpMinLatF = min(pnmm&lat)
   opcoes@mpMaxLatF = max(pnmm&lat)
   opcoes@mpMinLonF = min(pnmm&lon)
   opcoes@mpMaxLonF = max(pnmm&lon)
```

```
; plotando gráfico
plot = gsn_csm_contour_map( wks, pnmm, opcoes )

end
```

Neste script há um bloco de linhas de comando adicional que atribui coordenadas às dimensões do arranjo *pnmm*. Note a inclusão das unidades das coordenadas. Isto é importante. Caso não sejam definidas um erro ocorrerá e o gráfico não será gerado. Além disso, há cinco recursos adicionais, em relação ao script apresentado na seção 8.3. O recurso *gsnAddCyclic=False* avisa para a função **gsn_csm_contour_map** que os dados não são globais. Caso este recurso não tivesse sido definido desta maneira as isolinhas do campo não seriam plotadas.
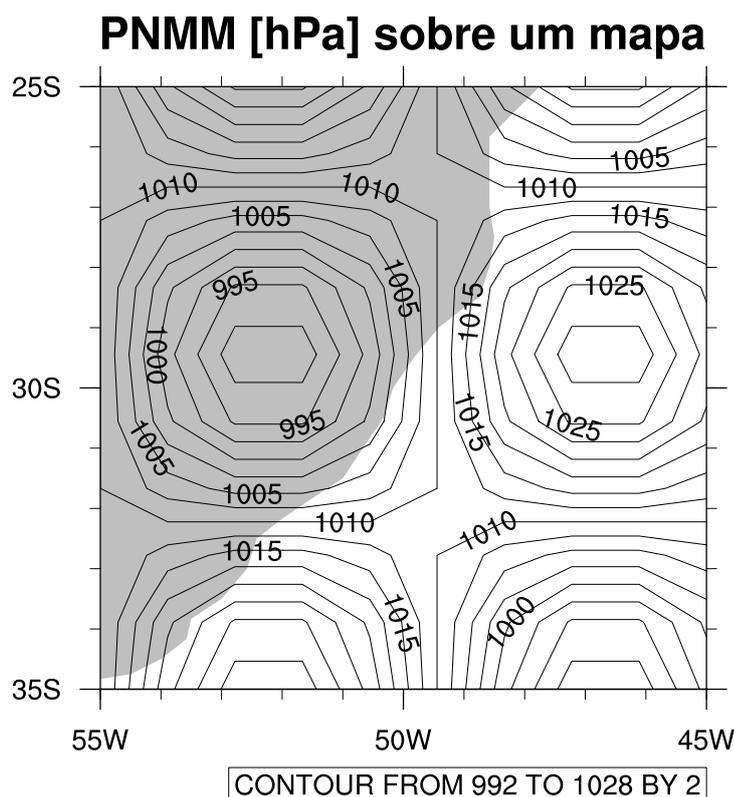
Figura 8.5: *Campo fictício de PNMM sobre um mapa. Veja o texto para maiores detalhes.*

Os recursos seguintes têm seus nomes iniciados pelos caracteres *mp*, indicando que são recursos gráficos que controlam a plotagem de mapas. Neste caso,

eles definem a área do mapa a ser plotado. Caso eles não tivessem sido definidos, os dados fictícios de PNMM apareceriam num mapa mundi. Note que os limites são definidos em função das coordenadas atribuídas ao arranjo *pnmm*. A Figura 8.5 exibe o resultado do script aqui apresentado e discutido. Veja que o mapa está preenchido e que não há divisões políticas. Há recursos gráficos para mudar isto. A página oficial do NCL oferece inúmeros exemplos que ilustram como alterar a plotagem de mapas. Visite-a!

# Apêndice A

# Acentuando palavras em gráficos do NCL

Não há no NCL nenhuma fonte com caracteres acentuados e nem a interpretação automática de strings com tais caracteres. Isso atrapalha os usuários cujas línguas nativas exigem estes caracteres. É possível escrever palavras com acentos no NCL, mas é um pouco chato.

Os caracteres acentuados são gerados por meio de Códigos de Função, que permite controlar vários aspectos de um texto. Veja a descrição completa destes códigos em `http://www.ncl.ucar.edu/Document/Graphics/function_code.shtml`.

Para facilitar a tarefa de escrever corretamente, apresentamos abaixo os códigos para gerar os caracteres acentuados. Os nomes das variáveis do tipo string, que contêm os códigos, seguem os nomes da tabela de acentos do HTML. Veja abaixo os caracteres acentuados e algumas palavras em português, mostrando como o NCL os plota.

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"

begin

; tabela de acentos
Agrave = "A~H-15V6F35~A~FV-6H3~" ; À
agrave = "a~H-13V2F35~A~FV-2H3~" ; à
Aacute = "A~H-15V6F35~B~FV-6H3~" ; Á
```

```
aacute = "a~H-13V2F35~B~FV-2H3~" ; á
Acirc = "A~H-15V6F35~C~FV-6H3~" ; Â
acirc = "a~H-13V2F35~C~FV-2H3~" ; â
Atilde = "A~H-15V6F35~D~FV-6H3~" ; Ã
atilde = "a~H-13V2F35~D~FV-2H3~" ; ã
Auml = "A~H-15V6F35~H~FV-6H3~" ; Ä
auml = "a~H-13V2F35~H~FV-2H3~" ; ä

Egrave = "E~H-15V6F35~A~FV-6H3~" ; È
egrave = "e~H-13V2F35~A~FV-2H3~" ; è
Eacute = "E~H-15V6F35~B~FV-6H3~" ; É
eacute = "e~H-13V2F35~B~FV-2H3~" ; é
Ecirc = "E~H-15V6F35~C~FV-6H3~" ; Ê
ecirc = "e~H-13V2F35~C~FV-2H3~" ; ê
Euml = "E~H-15V6F35~H~FV-6H3~" ; Ë
euml = "e~H-13V2F35~H~FV-2H3~" ; ë

Igrave = "I~H-10V6F35~A~FV-6H3~" ; Ì
igrave = "i~H-10V2F35~A~FV-2H3~" ; ì
Iacute = "I~H-08V6F35~B~FV-6H3~" ; Í
iacute = "i~H-08V2F35~B~FV-2~" ; í
Icirc = "I~H-09V6F35~C~FV-6H3~" ; Î
icirc = "i~H-09V2F35~C~FV-2H3~" ; î
Iuml = "I~H-09V6F35~H~FV-6H3~" ; Ï
iuml = "i~H-09V2F35~H~FV-2H3~" ; ï

Ograve = "O~H-15V6F35~A~FV-6H3~" ; Ò
ograve = "o~H-13V2F35~A~FV-2H3~" ; ò
Oacute = "O~H-15V6F35~B~FV-6H3~" ; Ó
oacute = "o~H-13V2F35~B~FV-2H3~" ; ó
Ocirc = "O~H-16V6F35~C~FV-6H3~" ; Ô
ocirc = "o~H-14V2F35~C~FV-2H3~" ; ô
Otilde = "O~H-15V6F35~D~FV-6H3~" ; Õ
otilde = "o~H-13V2F35~D~FV-2H3~" ; õ
Ouml = "O~H-16V6F35~H~FV-6H3~" ; Ä
ouml = "o~H-14V2F35~H~FV-2H3~" ; ä

Ugrave = "U~H-15V6F35~A~FV-6H3~" ; Ù
ugrave = "u~H-13V2F35~A~FV-2H3~" ; ù
Uacute = "U~H-13V6F35~B~FV-6H3~" ; Ú
uacute = "u~H-13V2F35~B~FV-2H3~" ; ú
```

```
Ucirc = "U~H-15V6F35~C~FV-6H3~" ; Û
ucirc = "u~H-13V2F35~C~FV-2H3~" ; û
Uuml = "U~H-15V6F35~H~FV-6H3~" ; Ü
uuml = "u~H-13V2F35~H~FV-2H3~" ; ü

Cedil = "C~H-15F35~K~FH2~" ; Ç
cedil = "c~H-13F35~K~FH2~" ; ç

Ntilde = "N~H-15V6F35~D~FV-6H3~" ; Ñ
ntilde = "n~H-13V2F35~D~FV-2H3~" ; ñ

; ambiente grafico
wks = gsn_open_wks("x11","acentos")

; recursos do texto
txres = True
txres@txFontHeightF = 0.03

; plotando caracteres acentuados
txres@txJust = "CenterCenter"
titulo1 = "Acentua"+cedil+atilde+"o com o NCL"
gsn_text_ndc(wks,titulo1,.5,.95,txres)

txres@txJust = "CenterLeft"
titulo2 = "Caracteres acentuados:"
gsn_text_ndc(wks,titulo2,0.,.85,txres)

texto = Agrave+" "+agrave+" "+Aacute+" "+\
aacute+" "+Acirc+" "+acirc+" "+\
Atilde+" "+atilde+" "+ \
Auml+" "+auml
gsn_text_ndc(wks,texto,.2,.8,txres)

texto = Egrave+" "+egrave+" "+Eacute+" "+\
eacute+" "+Ecirc+" "+ecirc+" "+\
Euml+" "+euml
gsn_text_ndc(wks,texto,.2,.75,txres)

texto = Igrave+" "+igrave+" "+Iacute+" "+\
iacute+" "+Icirc+" "+icirc+" "+\
Iuml+" "+iuml
```

```
gsn_text_ndc(wks,texto,.2,.7,txres)

texto = Ograve+" "+ograve+" "+Oacute+" "+\
oacute+" "+Ocirc+" "+ocirc+" "+\
Otilde+" "+otilde+" "+\
Ouml+" "+ouml
gsn_text_ndc(wks,texto,.2,.65,txres)

texto = Ugrave+" "+ugrave+" "+Uacute+" "+\
uacute+" "+Ucirc+" "+ucirc+" "+\
Uuml+" "+uuml
gsn_text_ndc(wks,texto,.2,.6,txres)

texto = Cedil+" "+cedil+" "+Ntilde+" "+ntilde
gsn_text_ndc(wks,texto,.2,.55,txres)

; algumas palavras acentuadas
titulo3 = "Algumas palavras:"
gsn_text_ndc(wks,titulo3,0.,.45,txres)

txres@txFontHeightF = 0.025
texto = "ver"+atilde+"o - inst"+aacute+\
"vel - hist"+oacute+"rico - mat"+eacute+"ria"
gsn_text_ndc(wks,texto,.1,.4,txres)

texto = "precipita"+cedil+atilde+\
"o - INFORMA"+Cedil+Otilde+"ES - tr"+ecirc+"s"
gsn_text_ndc(wks,texto,.1,.35,txres)

texto = "dire"+cedil+otilde+\
"es - El-Ni"+ntilde+"o - LA-NI"+Ntilde+"A"
gsn_text_ndc(wks,texto,.1,.3,txres)

texto = "got"+iacute+"culas - pol"+ecirc+\
"mica - Amaz"+ocirc+"nia"
gsn_text_ndc(wks,texto,.1,.25,txres)

texto = "mec"+acirc+"nica - portugu"+ecirc+\
"s - ci"+ecirc+"ncia atmosf"+eacute+"rica"
gsn_text_ndc(wks,texto,.1,.2,txres)
```

```
texto = Aacute+"reas - din"+acirc+"mica - Sa"+\
uacute+"de"
gsn_text_ndc(wks,texto,.1,.15,txres)

frame(wks)

end
```

Neste exemplo, os caracteres acentuados estão no script que os usam, mas podem ser colocados num arquivo .ncl separado que pode ser carregado com o comando **load**. A Figura A.1 apresenta o resultado do script.

Acentuação com o NCL

Caracteres acentuados:
À à Á á Â â Ã ã Ä ä
È è É é Ê ê Ë ë
Ì ì Í í Î î Ï ï
Ò ò Ó ó Ô ô Õ õ Ö ö
Ù ù Ú ú Û û Ü ü
Ç ç Ñ ñ

Algumas palavras:
verão - instável - histórico - matéria
precipitação - INFORMAÇÕES - três
direções - El-Niño - LA-NIÑA
gotículas - polêmica - Amazônia
mecânica - português - ciência atmosférica
Áreas - dinâmica - Saúde

Figura A.1: *Caracteres acentuados no NCL.*

A fonte de caracteres 35 contém os caracteres de acentuação, sendo usada, extensivamente, na criação dos caracteres acentuados. Vamos analisar a construção de um dos caracteres, sendo direta a extensão desta análise aos demais. Vejamos o caractere "á", definido no script acima como

aacute = "a~H-13V2F35~B~FV-2H3~"

A explicação se dará quase caractere por caractere:

| | |
|---|---|
| a | é o próprio caractere a, que será acentuado |
| ~ | é o código de função de texto, que pode ser modificado no arquivo .hluresfile. |
| H-13 | move a posição do texto atual 13 pontos na horizontal, à esquerda (negativo). |
| V2 | move a posição do texto atual 2 pontos na vertical, para cima. |
| F35 | usa a fonte de caracteres 35. |
| B | usa o caractere da fonte de caracteres 35. |
| F | volta a usar a fonte de caracteres anterior. |
| V-2 | move a posição do texto atual 2 pontos na vertical, para baixo. |
| H3 | move a posição do texto na horizontal, três pontos à direita. |

Em suma, os caracteres acentuados são montados manualmente, por meio de movimentos do cursor de impressão de caractere para a direita, esquerda, para cima e para baixo. Quem já usou uma máquina de escrever consegue percebeu o esquema de acentuação, ou seja, (i) escreve o caractere, (ii) volta o cursor para a posição do caractere escrito, (iii) eleva ou abaixa o cursor, (iv) escreve o sinal de acentuação e (v) faz o caminho de volta para continuar a escrever o texto.

Note que os códigos de função de texto ficam sempre entre ~, o símbolo indicador do uso destes códigos.

# Apêndice B

# Lendo arquivos ASCII complicados

É possível deparar-se com arquivos de dados com uma formatação complicada ou que mistura dados de tipos diferentes, por exemplo, uma tabela de dados contendo valores numéricos e strings de caracteres.

Suponha que exista um arquivo chamado "temp.txt" na pasta "dados", contendo colunas com as seguintes informações: latitude, longitude, temperatura do ar em 2 metros e o nome das estação meteorológicas separadas por ponto-e-vírgula. As primeiras linhas deste arquivo são mostradas abaixo:

```
-32.03;-52.09;32.8;Rio Grande
-31.87;-52.34;29.7;Pelotas
-30.54;-52.52;29.2;Encruzilhada do Sul
```

Para abrir o arquivo, usa-se a função **asciiread**. Entretanto, não é possível abri-lo usando esta função em seu modo habitual. Assim, os dados são lidos usando uma estratégia diferente:

```
abrir = asciiread("dados/temp.txt", -1, "string")
```

Note que não foram especificadas dimensões para os dados do arquivo ("-1") e o tipo de informação dentro do arquivo é definido como do tipo *string*, uma vez que existem letras e números. Caso este arquivo contivesse o código de cada estação, ao invés do seu nome, na quarta coluna dos dados, poderíamos abrir o arquivo com a linha de comando abaixo:

```
abrir = asciiread("dados/temp.txt", (/100,4/), "float")
```

Supondo, é claro, que os arquivo possua 100 linhas de dados. Voltando ao exemplo mais complicado, temos um arranjo unidimensional, chamado *abrir* contendo, em cada elemento, uma linha do arquivo de dados. Agora, para que cada coluna seja lida e armazenada em uma variável diferente, utilizamos a função **str_get_field**:

```
coluna1 = str_get_field(abrir, 1, ";")
coluna2 = str_get_field(abrir, 2, ";")
coluna3 = str_get_field(abrir, 3, ";")
coluna4 = str_get_field(abrir, 4, ";")
```

A função **str_get_field** usa o terceiro argumento passado a ela, neste caso o ponto-e-vírgula, para separar os dados em várias colunas. A partir desta separação, ele extrai da variável *abrir* a coluna indicada pelo número passado no segundo argumento. Agora, as variáveis *coluna1*, *coluna2*, *coluna3* e *coluna4* contêm, respectivamente, as informações de latitude, longitude, temperatura em 2 metros e o nome das estações. Entretanto, ainda não é possível usar as informações numéricas, pois elas estão como informações do tipo *string*. Para possibilitar o seu uso, vamos converter as informações para o formato numérico do tipo *float* usando a função **stringtofloat**:

```
lat = stringtofloat(campo1)
lon = stringtofloat(campo2)
t2m = stringtofloat(campo3)
```

Agora, é possível trabalhar com estas informações numéricas, ou seja, usá-las em cálculos e geoposicionamento. A linguagem NCL possui outras funções que transformam os campos de um tipo para outro, como **stringtoint** (transforma de *string* para *inteiro*), **strngtochar** (transforma de *string* para *caracter*) e **integertochar** (transforma de *inteiro* para *caracter*). Um guia completo destas funções pode ser obtido na página de documentação do ncl: `http://www.ncl.ucar.edu/Document/`.

# Apêndice C

# Divisão política em mapas

Nos estudos de meteorologia é comum a geração de mapas com campos atmosféricos, que permitem-nos analisar o tempo ou o clima para uma determinada região. Dependendo, é claro, do tipo do campo plotado. A localização geográfica das características observadas nos campos meteorológicos é crucial para a elaboração de diagnósticos e previsões.

Isto posto, percebe-se a importância de apresentar num mapa a separação política de uma região. Para ilustrar esta capacidade do NCL, vamos modificar o script apresentado ao final do Capítulo 8:

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin

   ; variaveis e parametros
   nx = 10
   ny = 10
   pi = acos(-1.)
   pnmm = new( (/ny,nx/), float )

   ; variacao dos eixos x e y
   dx = 2*pi/nx
   dy = 2*pi/ny

   ; gerando campo ficticio
```

```
    do j=0,ny-1
        do i=0,nx-1
            pnmm(j,i) = sin(i*dx)*cos(j*dy)*20.+1010.
        end do
    end do

    ; atribuindo coordenadas aos dados
    pnmm!0 = "lat"
    pnmm!1 = "lon"
    pnmm&lat = fspan(-35,-45,10)
    pnmm&lon = fspan(-55,-45,10)
    pnmm&lat@units = "degrees_north"
    pnmm&lon@units = "degrees_east"

    ; ambiente gráfico
    wks = gsn_open_wks( "x11", "campoFicticio")

    ; recursos gráficos
    opcoes = True
    opcoes@gsnMaximize = True
    opcoes@tiMainString = "PNMM [hPa] sobre um mapa  C  com divisao politica"
    opcoes@gsnAddCyclic = False
    opcoes@mpMinLatF = min(pnmm&lat)
    opcoes@mpMaxLatF = max(pnmm&lat)
    opcoes@mpMinLonF = min(pnmm&lon)
    opcoes@mpMaxLonF = max(pnmm&lon)

    ; recursos gráficos para o mapa
    opcoes@mpFillOn = False
    opcoes@mpDataSetName = "Earth..4"
    opcoes@mpDataBaseVersion = "MediumRes"
    opcoes@mpOutlineOn = True
    opcoes@mpOutlineSpecifiers = (/"Brazil:states"/)

    ; plotando gráfico
    plot = gsn_csm_contour_map( wks, pnmm, opcoes )

end
```

Note que foi incluído um novo bloco de linhas de comandos com novos recursos gráficos para mapas, ou seja, recursos cujos nomes iniciam com *mp*. O primeiro, *mpFillOn = False*, desliga o preenchimento do mapa, comportamento visto na plotagem de campos sobre mapas, no Capítulo 8. O segundo e o terceiro recursos gráficos, *mpDataSetName = "Earth..4"* e *mpDataBaseVersion = "MediumRes"*, definem o banco de dados de mapas a ser usado. Já o último, *mpOutlineSpecifiers = (/"Brazil:states"/)*, indicam que divisão deve ser plotada, que neste caso são os estados brasileiros. A Figura C.1 mostra o resultado deste script.
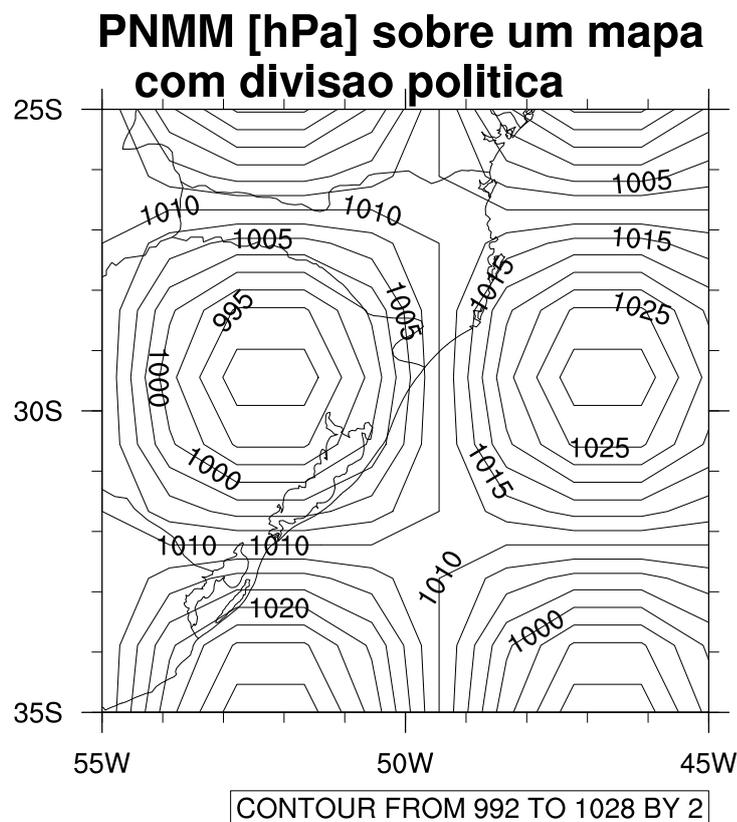
Figura C.1: *Campo fictício de PNMM sobre um mapa contendo a divisão estadual do Brasil. Veja o texto para maiores detalhes.*

Consulte os inúmeros exemplos relacionados a mapas no site oficial do NCL - `http://www.ncl.ucar.edu/Applications/maponly.shtml` - para conhecer as possibilidades oferecidas.

# Apêndice D

# Valores ausentes no NCL

Se uma variável do NCL tem um atributo chamado *_FillValue*, então todos os valores nesta variável que são iguais ao valor deste atributo são considerados como valores ausentes. Este atributo foi originalmente usado no formato de arquivo netCDF para representar dados ausentes e foi, posteriormente, adotado pelo NCL para o mesmo propósito.

Os exemplos abaixo mostram diferentes meios de se passar este atributo no NCL:

```
x@_FillValue = -999
x@_FillValue = default_fillvalue("double")
x = new( 5, double, 1e20 )
x@_FillValue = y@_FillValue
assignFillValue(y,x)
```

Na primeira linha, definimos o valor do atributo _FillValue como -999. Na linha seguinte, usamos a função **default_fillvalue** para definir o valor ausente padrão para variáveis tipo *double*. Na próxima linha, criamos uma nova variável, *x*, para a qual definimos como valor ausente 1e20. Na penúltima linha, passamos o valor do atributo _FillValue da variável *y* para o mesmo atributo da variável *x*. Por fim, a última linha faz a mesma coisa que a penúltima, mas usando a função **assignFillValue**.

Tabela D.1: Valores ausentes padrão para os vários tipos de variáveis do NCL. Estes valores são especificados para as versões anteriores e posteriores à versão 6.0.0, a qual trouxe mudanças importantes a eles.

| **Tipo da variável** | versões 5.2.x e anteriores | versões 6.x e posteriores |
|---|---:|---:|
| byte | 0xff | -127 |
| short | -99 | -32767 |
| ushort | 0 | 65535 |
| integer | -999 | -2147483647 |
| uint | 0 | 4294967295 |
| long | -9999 | -2147483647 |
| ulong | 0 | 4294967295 |
| int64 | -99999999 | -9223372036854775806 |
| uint64 | 0 | 18446744073709551614 |
| float | -999 | 9.96921e+36 |
| double | -9999 | 9.969209968386869e+36 |
| character | 0 | 0x00 |

## D.1 O uso de _FillValue em funções do NCL

Muitas funções do NCL ignoram automaticamente qualquer elemento cujo valor é igual ao do seu atributo *_FillValue*. O exemplo abaixo ilustra isto:

```
x = (/1., 2., 3., 4., 5./)
print(avg(x))

x@_FillValue = 5.
print(avg(x))
```

Neste exemplo, usamos a função **avg** para calcular a média aritmética dos elementos do arranjo unidimensional $x$, que no primeiro cálculo é igual a 3.0. Quando definimos o atributo *_FillValue* de $x$ como o valor 5.0, o novo cálculo da média ignora automaticamente o elemento cujo valor é igual a 5.0. Portanto, a média calculada agora é 2.5.

Muitas funções gráficas também reconhecem este atributo e não plotam dados que tenho valores iguais a ele.

## D.2 Valores ausentes padrão

Ao se criar uma variável no NCL e não se definir explicitamente o seu atributo *_FillValue*, o NCL define-o automaticamente usando valores padrão, que dependem do tipo de dados usado. A Tabela D.1 lista estes valores padrão.

## D.3 O atributo *missing_value*

É possível encontrar variáveis contendo um atributo chamado *missing_value*. O NCL não reconhecerá este atributo e, portanto, plotará e realizará cálculos com valores definidos como *missing_value*. Se queremos que o NCL reconheça estes valores como ausentes, é necessário renomear este atributo:

```
x@_FillValue = x@missing_value
delete(x@missing_value)
```

Sempre que é passado um novo valor ao atributo _FillValue, toda ocorrência de valores ausentes igual ao antigo valor deste atributo será alterada para o novo valor. Para ilustrar isso, vamos usar o vetor $x$ criado acima, para o qual passamos o valor 5.0 ao atributo *_FillValue*. Veja o que ocorre se redefinirmos o valor deste atributo:

```
x@_FillValue = 10.
print(x)

Variable: x
Type: float
Total Size: 20 bytes
            5 values
Number of Dimensions: 1
Dimensions and sizes: [5]
Coordinates:
Number Of Attributes: 1
 _FillValue : 10
(0) 1
(1) 2
(2) 3
(3) 4
(4) 10
```

Compare o conteúdo do vetor $x$ agora com o conteúdo inicial. É fácil notar que o último valor, que era igual a 5.0 foi alterado para 10.0.

## D.4 _FillValue = 0

Uma nota importante sobre a atribuição do valor 0 (zero) ao atributo *_FillValue*: é permitida esta atribuição, mas não é recomendada, uma vez que isto não funcionará para gráficos de contornos. Isto é um problema gráfico de baixo nível, ou seja, é um problema na programação do NCL. Portanto, evite passar a este atributo o valor 0 (zero).